

AD-A103 485

AIR FORCE LOGISTICS COMMAND WRIGHT-PATTERSON AFB OH  
PROCEEDINGS OF THE JOINT LOGISTICS COMMANDERS JOINT POLICY COOR--ETC(U)  
AUG 79

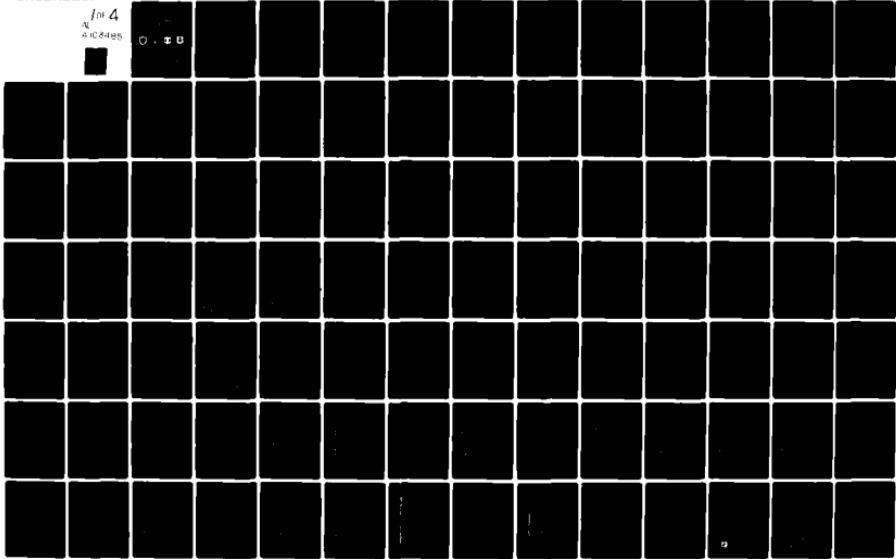
F/B 5/1

UNCLASSIFIED

NL

In 4  
AIC 64865

O - 20



ARMY MATERIEL DEVELOPMENT AND READINESS COMMAND  
NAVAL MATERIAL COMMAND  
AIR FORCE LOGISTICS COMMAND  
AIR FORCE SYSTEMS COMMAND

AD A 103485

PROCEEDINGS OF THE  
JOINT LOGISTICS COMMANDERS  
JOINT POLICY COORDINATING GROUP ON COMPUTER RESOURCE  
MANAGEMENT

COMPUTER SOFTWARE MANAGEMENT SUBGROUP  
SOFTWARE WORKSHOP

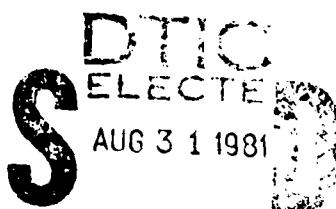


21 AUGUST 1979

"APPROVED FOR PUBLIC RELEASE - DISTRIBUTION  
UNLIMITED"

THIS REPORT IS NOT AN APPROVED JLC POSITION

DEPARTMENT OF THE ARMY, THE NAVY, AND THE AIR FORCE



A

80 8 28 057

"UNCLASSIFIED/UNLIMITED"

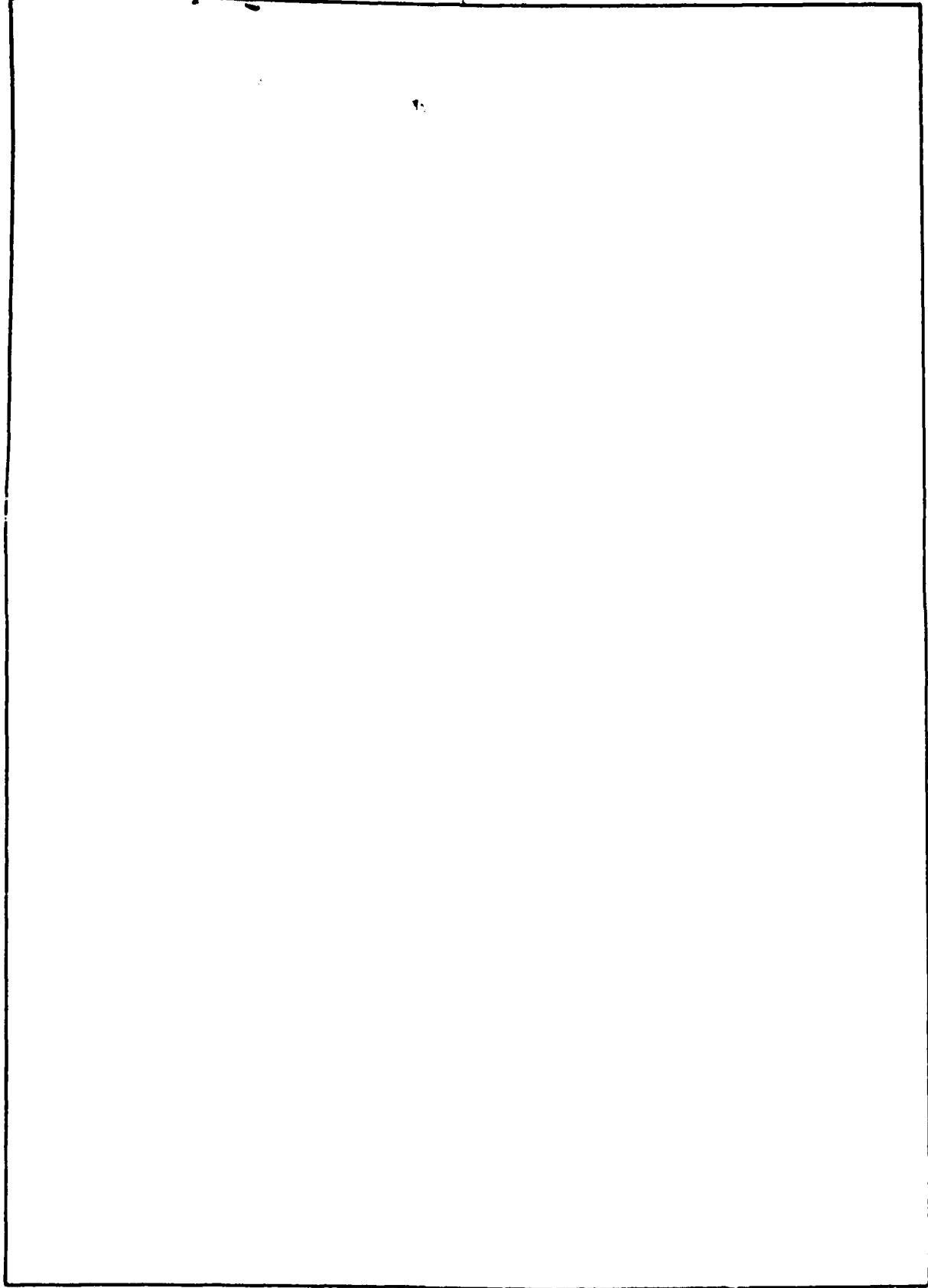
DMC FILE COPY

**UNCLASSIFIED**

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
AD-A103 485		
4. TITLE (and Subtitle) Proceedings of the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management; Computer Software Management Software Workshop 2-5 April 1979		5. TYPE OF REPORT & PERIOD COVERED Final
7. AUTHOR(s)		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS <i>Q</i>		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS HQ AFLC/LOEC Wright-Patterson AFB, OH 45433		12. REPORT DATE 21 August 1979
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES 346
		15. SECURITY CLASS. (of this report) <b>UNCLASSIFIED</b>
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved For Public Release, Distribution Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) These proceedings contain the Panel Chairpersons reports and Preliminary Panel Recommendations that were produced during the Joint Logistic Commanders Embedded Computer Software Workshop, 2-5 Apr 79 held at the Naval Postgraduate School, Monterey, CA.		

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

A large, empty rectangular box occupies most of the page, indicating a significant portion of the document has been redacted or removed.

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## TABLE OF CONTENTS

<u>Report of the Panel on Software Acquisition/Development Standards . . . . .</u>	1
Chairman's Cover Letter. . . . .	1
Objective . . . . .	3
Background and Scope. . . . .	3
Approach . . . . .	3
General Discussion. . . . .	4
Recommendations. . . . .	15
Appendix 1 - Panel Participants . . . . .	19
Appendix 2 - Bibliography . . . . .	27
Appendix 3 - Strawman-Embedded Computer System Software Development Life-Cycle Model Description. . . . .	31
Appendix 4 - Charts from Army/Navy/Air Force Briefing on Service Responses to DODD 5000.29. . . . .	35
Appendix 5 - Chart Integrating PDSS into Army Acquisition Policy . . . . .	91
 <u>Report of the Panel on Software Documentation . . . . .</u>	 93
Objective . . . . .	93
Scope. . . . .	93
Approach . . . . .	93
Discussion . . . . .	94
Recommendations. . . . .	96
Appendix 1 - Participants . . . . .	99
Appendix 2 - Software Documentation References. . . . .	101
Appendix 3 - Descriptive Paragraphs. . . . .	103
 <u>Report of the Panel on Standards for Software Quality. . . . .</u>	 127
Objectives. . . . .	127
Scope. . . . .	127
Approach . . . . .	129
Discussion . . . . .	131
Recommendations. . . . .	138
Appendix 1 - Participants . . . . .	141
Appendix 2 - Bibliography . . . . .	145
Appendix 3 - Military Specification Software Quality Assurance Program Requirements . . . . .	147
Appendix 4 - Software Quality Assurance Plan. . . . .	157
Appendix 5 - Comments on Panel C Report . . . . .	159

TABLE OF CONTENTS (Cont'd)

<u>Report of the Panel on Software Acceptance Criteria</u> . . . . .	161
Objective . . . . .	161
Scope . . . . .	161
Approach . . . . .	161
Discussion . . . . .	162
Recommendations . . . . .	183
Appendix 1 - Participants . . . . .	185
Appendix 2 - Bibliography . . . . .	187
Appendix 3 - Letter to Panel Participants. . . . .	189
Appendix 4 - Panel Handout . . . . .	191
Appendix 5 - Panel Presentation . . . . .	197
Appendix 6 - Visual Aids of Panel Presentation to Workshop . . . . .	233
Appendix 7 - Draft of Proposed Triservice Policy for Software Acceptance Criteria . . . . .	267
Appendix 8 - Outline and Draft Material for Proposed Triservice Interim Guideline on Application of Software Acceptance Criteria . . . . .	269
Appendix 9 - Addendum . . . . .	293

## LIST OF FIGURES

## Report of the Panel on Software Acquisition/Development Standards

<b>Figure 1.</b>	<b>Embedded Computer System Software Acquisition</b>	
	<b>Life-Cycle Model . . . . .</b>	<b>6</b>
<b>Figure 2.</b>	<b>Standards Used in Implementing Software Life-Cycle . . . . .</b>	<b>7</b>
<b>Figure 3.</b>	<b>Relationship of MIL Standards to Software Acquisition</b>	
	<b>Products and Activities . . . . .</b>	<b>9</b>
<b>Figure 4.</b>	<b>Ideal System Life-Cycle . . . . .</b>	<b>12</b>
<b>Figure 3-1.</b>	<b>Embedded Computer System Software Acquisition Life-Cycle</b>	
	<b>Model . . . . .</b>	<b>33</b>

## Report of the Panel on Software Acceptance Criteria

<b>Figure 1.</b>	<b>Sample System With Embedded Software. . . . .</b>	<b>165</b>
<b>Figure 2.</b>	<b>Error Model . . . . .</b>	<b>176</b>
<b>Figure 3.</b>	<b>Error Severity/Cost Matrices . . . . .</b>	<b>179</b>

Special  
Offer

## LIST OF TABLES

### Report of the Panel on Software Acceptance Criteria

Table 1.	DOD Software Standards . . . . .	168
Table 2.	Ideal Software Acquisition Cycle . . . . .	169
Table 3.	Correlation of Standards and Ideal Cycle . . . . .	170
Table 4.	Software Error Categories. . . . .	175
Table 8-1.	Ideal Software Acquisition Cycle . . . . .	271
Table 8-2.	Correlation of Standards and Ideal Cycle . . . . .	272
Table 8-3.	System Level Specification Set (FCI) Acceptance Criteria . . . . .	277
Table 8-4.	Software Acceptance Criteria for Development Spec and Program Requirements Spec . . . . .	278
Table 8-5.	Software Acceptance Criteria for Design Documentation (Product Specs) . . . . .	280
Table 8-6.	Software Acceptance Criteria (SAC) for the CPC (CODE) . . . . .	282
Table 8-7.	SAC for Test Plan Documents. . . . .	284
Table 8-8.	Software Acceptance Criteria for Test Procedures. . . . .	285
Table 8-9.	Software Test Reports' Software Acceptance Criteria . . . . .	286
Table 8-10.	Software Acceptance Criteria for Operators' and Users' Manuals . . . . .	287
Table 8-11.	Failure Severity/Cost Matrix . . . . .	290
Table 8-12.	Acquisition Impact/Intermediate Factor Matrix . . . . .	290

PROCEEDINGS OF THE SOFTWARE WORKSHOP  
JOINT LOGISTICS COMMANDERS'  
JOINT POLICY COORDINATING GROUP ON COMPUTER RESOURCE MANAGEMENT  
MONTEREY, CA, 2-5 APRIL 1979

Report of the Panel on  
Software Acquisition/Development Standards

5 April 1979

Chairman

John B. Munson - System Development Corporation

Co-Chairmen

Norman Berman - U.S. General Accounting Office

Robert Berri - Aerospace Corporation

**System  
Development  
Corporation**

2500 COLORADO AVENUE • SANTA MONICA, CALIFORNIA 90406  
TELEPHONE (213) 829-7511

TELEX 65-2358

June 1, 1979

**TO: Panel Members - Software Acquisition/Development Standards Panel**

Enclosed is the complete, final report to the Joint Logistics Commanders' Computer Resource Management Group from our panel on Software Acquisition/Development Standards.

I have included all comments received, except where I've spoken to you personally. Sorry there isn't time for one more review pass, but I believe the comments incorporated have only strengthened and amplified the report, not changed it in any substantive way.

I am pleased with the report, and I believe it is constructive on a very difficult subject. I thank you for your help and support.

As a result of our report, we may be asked to participate in some further tasks and activities. Please let me know if you would be willing and able to continue in this effort if we're asked.

I look forward to seeing and working with you again.



John B. Munson  
Vice President  
Corporate Software Engineering

JBM:jb  
Enclosure

## OBJECTIVE

Evaluate the potential for developing triservice standards for the acquisition of embedded computer system software. Determine the role of MIL Standard 1679 (Navy) within a set of triservice software acquisition standards.

## BACKGROUND AND SCOPE

Each service acquires software for embedded computer systems (ECS) within a very general framework as typified by MIL Standards 483, 490, and 1521. However, specific implementations vary widely between services and even between contracting agencies within the same service. Such items as documentation standards, configuration management and reviews, and audits are characterized more by their inconsistent implementation than by their universality. Additionally, such standards are only meant to be used within an overall framework (service policy) for software acquisition, called a "Software Acquisition Life-Cycle," which relates specific events, tasks, products, and reviews to a time-sequenced master plan for software development activities.

Over the years since the initial standards for software acquisition were developed, experience gained in their use has shown the need for updating and expanding them to account for experience, earlier errors and omissions, and improved techniques of software engineering, and to provide better visibility into the development process. DOD Directive 5000.29, published in April 1976, emphasized the importance of software in embedded computer system acquisitions and required each service to define its ECS software acquisition policy and to modify or add MIL Standards necessary to implement that policy.

The Software Acquisition Standards Panel looked at how each service responded to DODD 5000.29, evaluated the current MIL Standard structure to support the policy, hypothesized a common software acquisition life-cycle process, identified possible areas where current MIL Standards were inadequate or lacking, and evaluated the potential impact of a joint, triservice framework for software acquisition. Following this activity, MIL Standard 1679 (Navy) was evaluated for its compatibility as a complementary triservice standard.

## APPROACH

The panel on Software Acquisition Standards, consisting of 24 service and industry members (Appendix 1) met for 2 1/2 days, April 3-5, 1979. The first half-day was spent in briefings by panel members concerning how each service responded to DODD 5000.29\*. These briefings are attached as Appendix 4.

\* In summary, Air Force Reg. 800-14 is published and implemented. Army Reg. 70-XX and Navy Reg. 5200.XX are still drafts undergoing final review. All three regulations deal with the issue of implementing DODD 5000.29 and all specify general characteristics (albeit, varying widely in level of detail) for implementing specific software acquisition practices.

Following the briefings, the panel divided into three working groups to investigate three major sub-issues:

1. Potential commonality (triservice) for software acquisition life-cycle management (J. Munson, Chairman).
2. Adequacy of current MIL Standards to support common acquisition management, including MIL Standard 1679 (Navy) (R. Berri, Chairman).
3. Adequacy of current acquisition practices relative to acquiring capabilities needed for post-development/deployment software support (PDSS) activities (e.g., operations and support) (N. Berinan, Chairman).

During the remaining 2 days, each working group independently pursued their goals and held joint coordination meetings twice daily, leading to a consolidated consensus summary and report containing five specific areas of recommendation to the full computer resource management group the afternoon of 5 April.

#### GENERAL DISCUSSION

Four major observations became apparent to the panel in the first joint session and these drove most of the activities and discussion for the rest of the investigations. These were:

1. Specific MIL Standards can be discussed and evaluated only in the context of an existing, overlying software acquisition life-cycle policy.
2. No technical reason for not having a consistent (triservice) MIL Standard framework was seen, nor could any be hypothesized.
3. Portions of MIL Standard 1679 (Navy) definitely fill a void in current acquisition practices; but the Standard is written in such a way that it is "stand-alone" regarding existing standards and practices.
4. Most current emphasis on software acquisition policy considers software development as a discrete activity and fails to adequately integrate its acquisition into the broader system acquisition framework (in contrast with the intent of DOD 5000.29).

## Software Acquisition Life-Cycle Policy

Any discussion of the adequacy of a specific MIL Standard for use in software acquisition management immediately raises the issue that standards are specific, discrete tools used in the more general process of specifying and managing the development of software for embedded computer systems. As such, a meaningful evaluation of standards can be made only in light of a more general policy which defines a software acquisition framework and includes the tasks, events, products, activities, and controls which will be employed as an overlying strategy for acquiring software. This framework, generally called a software acquisition development life-cycle model, provides the context in which standards can describe the various specific contractual requirements for successfully implementing a software development activity (Figure 1).

Each service, to a greater or lesser degree, has implemented its policy to provide this basic framework. The panel found that to the level documented, all services were basically in accord at the policy level. However, differences in both nomenclature and emphasis tended to confuse the interpretation, while variations in level of detail made it difficult to be sure how any given acquisition agency would interpret these instructions. For the panel's purposes, only by constructing a detailed, composite life-cycle model, based on the needs of a hypothetical ECS software RFP, was it possible to structure a basis for evaluating discrete standards. This led the panel to the conclusion that either each service policy should be reviewed by the JLC to ensure that each is detailed, complete, and consistent in the area of life-cycle management or to determine that a single policy should exist for all services. In the panel's opinion, that area of service regulations (i.e., the Regulations AR70-XX, AFR 800-14, and SECNAVINST 5200.XX) dealing with ECS software acquisition (excluding organizational roles, missions, and relationships) should be consolidated into one joint service policy statement. Without such a common framework and implementation directive, the development of proposed joint standards becomes problematical at best. The panel summarized the work it went through to develop a life-cycle model, and has included it as Appendix 3. The panel believes this strawman model can, and should, serve as the basis for constructing a joint-service policy statement on ECS software acquisition management.

## Triservice MIL Standards

In evaluating specific standards to implement the proposed life-cycle model, the panel used the information supplied by General Lasher, Figure 2, service-specific documentation, the references listed in Appendix 2, and the broad technical knowledge of the individual panel members. The panel was also well-informed of the efforts of the three other Computer Resource Management (CRM) panels, which were reviewing software documentation, quality assurance, and acceptance criteria.

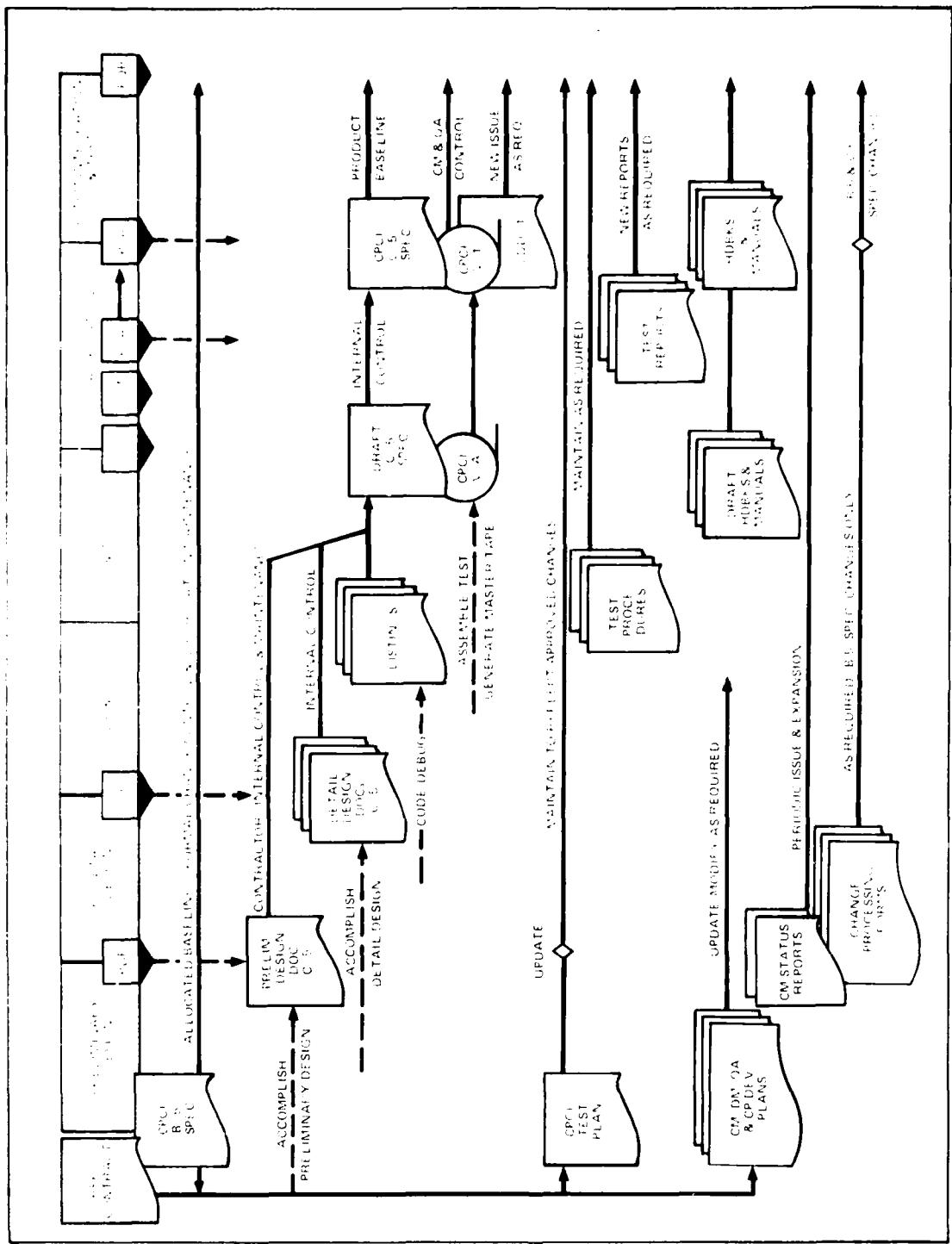


Figure 1. Embedded Computer System Software Acquisition Life-Cycle Model

STANDARD	SOFTWARE STANDARDS										DATA BASE	
	MILITARY STANDARDS (MIL STD.S.)					REQUIREMENTS			CODES		TESTING	
DOD INSTRUCTION MANUALS		SPECIFICATIONS		DESIGN		DELIVERY		INSTALLATION		CONFIGURATION MANAGEMENT		
MILITARY STANDARDS (MIL STD.S.)	785* 1521	490 143 5-83490	756A* 889* 1472	483	109*	480 483 482 785A* 481	0 9858A* 5 52779	483 100 490	756A* 757 785* 785	482 721 109	483	483
ARMY	1000-1					70 10 71 3 1000 1				70 10	310 1	70 10
NAVY	1679	1679	1679	1679	1			4130 1			1679	
AIR FORCE	800-14					80 14	800 14	65 3 57 4	122 9	122 9	800 14	
COMMERCIAL STANDARDS (ANSI, IEEE ASA, USASC II)								N41 3 338 1'		N41 4*	C 16 35 352 1972	FIPS 38 FIPS 30
										3 12	1970	FIPS 29 FIPS 30
										162	1963	
										163	1959	FIPS 11

\*APPLIES GENERALLY, NOT SPECIFICALLY, TO SOFTWARE, OR IS AMBIGUOUS WITH RESPECT TO SOFTWARE.

Figure 2. Standards Used in Implementing Software Life-Cycle

To summarize this evaluation, the panel generated Figure 3, which characterizes software acquisition as products, activities, and controls, and relates existing MIL Standards to each element. As noted in the chart, key elements are currently covered by existing MIL Standards, with the exception of "development activities."

Those specific areas with the highest priority need for review and improvement are currently being addressed by the JLC-CRM (e.g., specifications and documentation, QA, and acceptance criteria). Other applicable standards were identified as needing additional work, but were not deemed to be as high priority as the ones currently being addressed by the JLC-CRM. Since the only tangible product of software development is documentation (from requirements specifications to computer program listings and test reports), the area of specifications and other documentation is key to the entire acquisition process, and therefore is the basis for the life-cycle model framework and the highest priority project.

In essence, the panel's conclusion from this review is:

1. Triservice commonality of military software acquisition standards is possible, practical, and essential to rational military software acquisition management and industry's cost-effective ability to respond to it.
2. This commonality should be achieved within a common triservice ECS software acquisition life-cycle development policy.
3. This commonality should be built on the existing framework currently utilized in most military agencies today (MIL Standards 483, 490, 1521, 52779, etc.).

#### MIL Standard 1679 (Navy)

This conclusion still leaves the issue of MIL Standard 1679 (Navy). In this case, the panel concluded that MIL Standard 1679 (Navy) contains many concepts which are important additions to acquisition policy and should be tailored to address those activities (identified in Figure 3) which deal specifically with software production, currently the major unaddressed area. Also, the new MIL Standard 1679 (Navy) should recognize existing MIL Standards for those areas where adequate standards exist or are being generated. Specifically, Sections 5.3 through 5.8 of MIL Standard 1679 (Navy) should be used as the basis for a triservice software production standard which can specify and control production practices of software development organizations. This conclusion recognizes the current, sharply divided opinion relative to the advisability of a standard dealing with what until now have been contractor-discretionary production management practices.

SOFTWARE ACQUISITION MANAGEMENT ELEMENTS			
PRODUCTS	ACTIVITIES	CONTROLS	
PLANS(3)		REVIEWS AND AUDITS – 1521	
SPECIFICATIONS 483, 490	ANALYSIS DESIGN CODING	CM/DM – 480, 483, 490(4) QA – 52779(1)	
DOCUMENTATION – DATA ITEMS	INTEGRATION CONTRACTOR TEST	I/F CONTROL – 490(1) (MANAGEMENT CONTROLS)(5) 881(2)	
TERMS AND DEFINITIONS(3)			
PRODUCT MEDIA – DATA ITEMS	FORMAL TEST(3)		
DATA BASE – 490(1)	ACCEPTANCE – UNDER DEVELOPMENT		

NOTES:

- (1) MIL Standards 483, 490 and 52779 are currently being revised by JLC-CRM. Any new issue should include provision for data base and interface control.
- (2) 881 should be revised to emphasize software, increase software to level 3 and incorporate a "model" WBS format.
- (3) Should be defined in triservice regulation.
- (4) Needs major revision for software CM and DM.
- (5) Most MIL Standards for management control are system-level, and they are not considered in this review.

Figure 3. Relationship of MIL Standards to Software Acquisition Products and Activities

In the panel's opinion, the revision of MIL Standard 1679 (Navy) must take into account the technical work currently being done on the same subject by the Air Force's Electronic Systems Division and by the Rome Air Development Center, as well as studies done by the Air Force Test & Evaluation Command concerning software maintainability characteristics.

#### Post-Development/Deployment Software Support

In the review of the impact of post-development/deployment software support (PDSS) on the acquisition cycle, the panel recognizes that the various services have tried to strengthen front-end planning to more adequately address this issue in the future. The Army's Computer Resource Management Plan (CRMP) and the Air Force CRISP both deal with the planning necessary for PDSS. The review of the requirements expressed in these plans identified several major areas of shortcomings in respect to PDSS which will clearly impact, and could cause failures in, future acquisitions. One of our panel members has recently performed an Army study of the life-cycle events as they relate to PDSS. This study is included in Appendix 5 as an example of how pervasive this front-end planning must be to achieve an adequate software logistics capability. This concern for software logistics must be expanded beyond the scope of what either the CRMP or CRISP currently contemplate and must be defined much as the Navy LCMP is defined in 5200.23. These plans, developed in concert with the intended PDSS support agency, must address the support concept and relate the needs of the support organization in acquisition and development plans. For example, such items as the development of computer programs for data simulation, test data recording and reduction techniques, and built-in test diagnostics must be procured and validated with the operational programs. Furthermore, provision to buy adequate maintenance documentation, the required delivery of contractor-developed (or even proprietary) development tools, and even ensuring delivery of all test materials and results (for use in PDSS regression testing) may be essential to cost-effective PDSS and must be contemplated in the development contract. In addition, any software development techniques which can improve the characteristics for enhanced software PDSS must either be added to the revised MIL Standard 1679 (Navy) or be included in the statement of work at development time.

Finally, a software development model for PDSS should be developed to ensure that the disciplines and controls enforced during acquisition are not totally disregarded during the activities of error correction and program modification in PDSS.

#### The Need to Embed Software Acquisition into System Acquisition

The panel addressed the issue that although the JLC was taking important and significant steps to improve our military's capability to acquire software (either stand-alone software or software embedded in systems), little has been done to relate these procedures to system procurement philosophy. Also, little authoritative guidance exists to aid acquisition agencies building an embedded computer system concerning how the total system can be procured and managed relative to the software. These very important issues can be addressed and explained best by some background followed by some examples.

Although the system life-cycle is generally initiated by an ROC (Required Operational Capability) or equivalent statement of the user requirement, and the system is defined by an "A" level specification (see Figure 4), the software acquisition cycle is essentially concerned with the full-scale development phase or its equivalent. It starts with an authenticated (approved) B-5 specification (performance requirements) and ends with a Formal Qualification Test (FQT), which ensures that the delivered computer program performs as specified in the B-5 specification. On the other hand, procurement of an embedded computer system starts back in the conceptual phase and ends with transfer and turnover of the system to the using command. Figure 4 is a simplified description of these events. The first issue is how an acquisition agency gets to the point where it can accomplish software procurement. The most straightforward approach is to go through a typical validation phase, which develops configuration item specification (B-level) for each element of the system; then, the agency competes and contracts for CIs in accordance with normal acquisition procedures. However, the responsibility for the system (as opposed to subsystems) belongs to the military acquisition agency—not the CI contractors. If incompatibilities between B-level specifications and A-level specifications (system performance) occur during test, the acquisition agency must pay in money and time to have these deviations resolved.

Another alternative to solve this problem is to have the validation phase (B-level specification development) contracted for in a competitive validation phase (more than one contractor in a "shoot-out" leading to selection of a winning contractor to handle full-scale development, the selection being based on the quality and cost associated with the B-level specifications). However, there is little or no formal acquisition guidance for a competitive validation phase. There is no provision to take the best work and ideas from all contractors and combine them without a new, open competition at the start of full-scale development. After this long process, when the acquisition agency finally contracts for the CIs in accordance with these B-level specifications, the government is still responsible for the total system performance. While this method is used frequently today, it seems to work best when the acquisition agency hires an integrating contractor to put the pieces together.

Finally, the acquisition agency can contract with a "prime contractor" on the basis of the A-level (system performance) specification, making the prime contractor responsible for total system performance. In this case, the government has little or no control over how CIs are defined, and must depend on the prime to enforce CI-level acquisition management procedures, including software. All the controls and MIL Standards, developed so that the government can monitor and control product development (i.e., B-5 specifications, PDRs, CDRs, and FQTs), in effect, become just advisory to them; and they are now under management control of the prime, who is contracted to the Government for an A-level specification and who will officially deliver in accordance therewith many years later, following system integration and development test and evaluation (DT&E). Implementation of embedded computer systems, specifically C<sup>3</sup>I systems, has been extremely confused in recent years in this entire area.

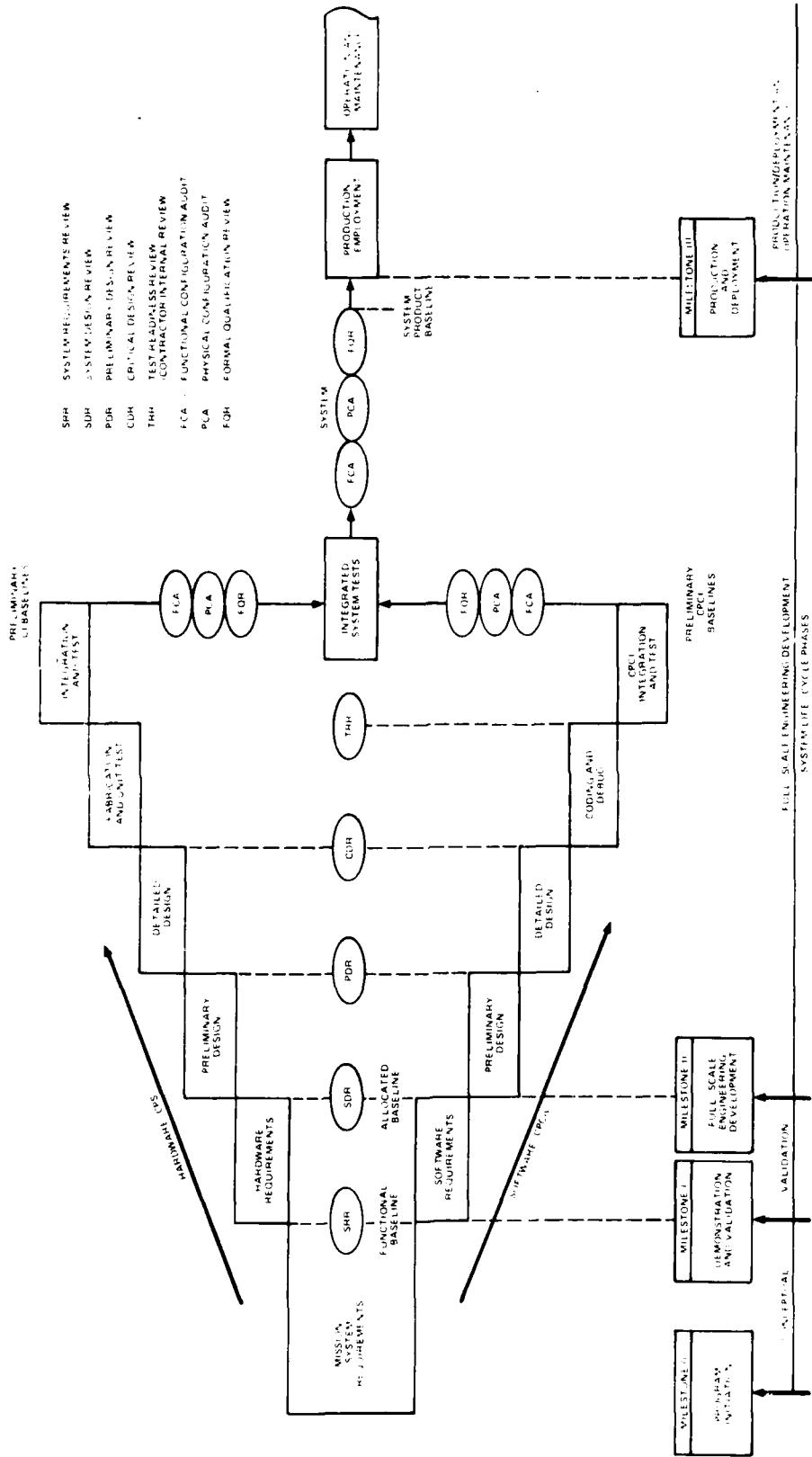


Figure 4. Ideal System Life-Cycle

For example, if the Government is involved with the prime in review and authentication of both A- and B-level specifications, against which set should the contractor's performance be accounted? Who runs configuration management and chairs the CCB (and is accountable for costs due to change)?

Another issue arises during test and acceptance and is the mirror image of the specification problem. If the services acquire CIs, then they have to perform system-level integration (or contract for an integrator). Likewise, if they use a prime, the Government cannot use FQT as formal qualification tests, but instead must wait until contractor DT&E is completed.

All in all, this issue basically resolves into control and accountability problems that are currently not well defined, especially regarding software and how it is to be controlled in total system acquisition.

The panel has no major conclusion or recommendation as a result of this discussion. In fact, the panel's charter did not include this issue, except the need to identify it as a major unresolved issue, potentially impacting the JLC efforts to bring software acquisition management into uniform structure. The general assessment was, however, that the system-level issues in the final analysis may have a greater impact on software success than the software acquisition issues.

#### Other Discussion

During the course of the panel's deliberations, a number of minor sub-issues were explored and are recorded here only as information:

1. **Firmware**—The panel felt firmware was only software delivered in a unique media and should be controlled by the same procedures used for software acquisition. Furthermore, if anything, even more rigor, especially during test, should be employed to assure the correctness of firmware because of the relative inflexibility of the media and the higher likelihood of multicopy production.
2. **Microprocessors/Microcomputers**—These were considered by all a major problem in future systems—especially with the high likelihood of their being very deeply embedded into systems (e.g., within hardware CIs), and the high probability that they will not exist in systems as conventional computers (i.e., no peripherals and limited hard-wired I/O channels), thus precluding an ability for individual test and qualification.
3. **PASCAL for Microcomputers**—PASCAL is becoming a de facto industry standard (or lack thereof) for micros and there has been no recognition of this fact in DODD 5000.31. In addition, it seems clear that commercial pressures will tend to drive microprocessor/computer technology rather than military controls or needs. At this point, everyone appears to be ignoring the problem and waiting for a crisis to focus attention on the issue.

4. Operational Test—The ability for a contractor to test major embedded computer systems in anything close to a realistic operational environment is virtually impossible in today's world. The implication is that many systems reach deployment before their true operational effectiveness has been measured.
5. Independent "V&V" (Verification and Validation)—V&V has become a major new initiative today as a methodology for improving software quality. However, there is no agreed-upon definition for V&V or how it is contracted.

The panel concluded that V&V is basically the practice of using a separate contractor (from the software development contractor) to do those technical tasks the customer would do if he had the requisite technical manpower in-house. These tasks subdivide into four major activities:

- a. Design verification through independent review, studies, modeling, and technical analysis.
- b. Product development traceability:
  - (1) Trace B-level specification to A-level specification requirements.
  - (2) Trace C-level specification to B-level specification requirements.
  - (3) Trace Section 4 (test) of B-level specification (CPCI) to CPCI test procedures (test specification for Navy).
    - For Navy, trace test specification to test procedures.
  - (4) Trace Section 4 of A-specification to system test procedures.
  - (5) Identify or execute added testing.
  - (6) Witness testing for Government when required.
- c. Technical review of contractor development activities.
- d. Validation of test results through analysis, separate testing, and contractor test monitoring.

This activity can be extremely valuable to an acquisition agency, but it can also be very expensive (20-30 percent of development cost). It must be used selectively and it must be carefully contracted for to prevent contractual clashes between the V&V contractor and the development contractor.

6. "Preliminary Design" (a note for the Specifications and Documentation Panel)—MIL Standards 1521, 483, 490, and AFR 800-14 all recognize the need for an early, formal review of the evolving CPCl design. Provision for this review is envisioned at PDR, when the initial design studies and CPCl system architecture are reviewed and compared against the authenticated, approved, and baselined B-5 specification. Unfortunately, current specification standards do not envision a single document to collect, structure, and record this information, nor is a consistent level of detail defined. In practice, PDR can range from a review and rehash of the B-5 specification to a dog-and-pony show to educate (or overwhelm) the customer representatives.

We propose that either a new document be created or the C-5 specification be defined in two volumes, so that a deliverable system architecture can be documented and reviewed prior to its formal presentation at PDR. This document would record all software system design down to the CPC level and would contain all system performance—time, space, and accuracy—budgets for the total software system, and a record of how these are allocated to major segments of the software architecture. In addition, this document would present CPCl control structure, data base organization, interface provisions, test plans, and support tools. At PDR this document would be reviewed and verified (i.e., compared to the B-5 specification) with design and trade studies presented to support selection of the candidate architecture.

Following PDR, the contractor would maintain this document, keeping it current and using it to support the detailed design presentations at CDR. This document, plus the C-5 (Vol. II) would become the product specification following approval at PCA.

## RECOMMENDATIONS

### 1. Issue: Establishment of Triservice Software Acquisition Standards

#### a. Conclusion

The panel could find no technical reason why consistent software acquisition standards cannot be established as triservice policy today.

#### b. Recommendation

Continue the JLC effort to revise and modernize existing MIL Standards for software acquisition based on the workshop's efforts, the preliminary prioritization contained herein (Figure 3), and a final prioritization to be established by the Software Management Subgroup of the JLC-JPCG-CRM.

2. Issue: How Does MIL Standard 1679 (Navy) Fit Into a Triservice Acquisition Policy?

a. Conclusion

The concepts of MIL Standards 1679 (Navy) can fill an important but neglected role in a comprehensive set of triservice standards. However, MIL Standard 1679 (Navy) must be modified and tailored to fit the other proposed standards.

b. Recommendation

Use MIL Standard 1679 (Navy) as the basis for creating a new triservice standard which deals with software development practices. This requires that the Navy MIL Standard be tailored to mesh with existing standards and procedures and be augmented by other existing service efforts in the area of standards for development practices.

3. Issue: Establishment of Triservice Software Acquisition Policy.

a. Conclusion

The panel considers it essential to the implementation of coherent software acquisition policy that the JLC formulate and implement a triservice regulation on software acquisition management that defines a consistent approach to implementing MIL Standards in the acquisition process.

b. Recommendation

Those areas of the service policy responses to DOD Directive 5000.29 should be incorporated into a triservice regulation. To accomplish this, the JCL's JPCG-CRM should establish a new working group chartered to:

- (1) Define a triservice regulation covering the common functional elements and milestones which comprise the service's software acquisition life-cycle. These functional elements should be defined to the level of detail which describes how they can be implemented by the system user, developer, maintainer, and other operational support agencies.

It is further recommended that AFR 800-14 be used as a basis for developing such a regulation since it is the most fully developed of the existing service regulations. See Appendix 3 for a strawman version.

- (2) Identify the service documents/policies which are aimed at implementation of similar functions (for example, PDSS) and arrive at a common title for both the activity and the document which guides the implementation of each function.
- (3) Coordinate the service's evolutionary efforts to achieve commonality among the supporting documents which address similar functions.

4. Issue: Weapon System Acquisition Policy as it Impacts Software Acquisition.

a. Conclusion

Military acquisition policy today is oriented towards procurement of contract end items. Yet, with the complexity of modern computerized systems, the services have needed to acquire these as total, integrated systems. Adequate acquisition policy and guidance does not currently exist relating to how software is procured when it is a portion of a larger system, or when it is being managed by a prime contractor rather than the Government.

b. Recommendation

The JLC-CRM should evaluate the implications for the acquisition policy relating to software when it is included as part of a larger system procurement.

5. Issue: Post-Development Operational Software Support.

a. Conclusion

The needs of the post-development software support activities are being treated neither early enough in the life-cycle nor thoroughly enough regarding issues which must be resolved during software acquisition.

b. Recommendation

- (1) The panel believes that sufficient guidance already exists throughout the services to identify the important life-cycle planning factors relating to post-development support; however, the guidance should be consolidated and enforced.
- (2) The panel believes that the requirements for post-development support resources should be addressed early in the development process for every weapon system. The post-deployment support organization should be identified prior to the planning for engineering development so that the support organization can participate in that planning, and particularly in the early definition of post-development resources and facilities.

- (3) The panel believes that the process for acquiring a quality support capability (which may include software and/or hardware) needs improvement, especially with regard to factoring these needs into ECS procurements:
  - (a) The post-development support agent should participate in verifying acceptability of support technical resources. This process is an important factor in assessing system maintainability. It appears that this is not uniformly done as part of acceptance or the operational testing process.
  - (b) Project offices lack sufficient information on the availability of existing software support tools and facilities. This information is needed before a decision is made to contract for development of new support resources.
  - (c) Project managers should ensure that support resources acquired during system development can be efficiently transferred to the support agency by replication or direct transfer. This transfer may include: required use of Government-furnished software; use of existing designs; and compatibility with existing government facilities.
  - (d) System-peculiar automatic test equipment is wrongly being categorized as a nonmission essential support item. Instead, it should be developed and tested as part of the weapon system.
  - (e) Specific funds to acquire adequate support resources should be earmarked by each project office when it formulates the system cost baseline for engineering development.
- (4) Current development standards do not adequately define the test and evaluation process and level of test requirements for software changes made during the post-deployment support phase. There is a need to first define how to classify changes according to significance. Then, the services should define the process, including test and evaluation, by which software is released for service use.
- (5) The panel has been advised that the Air Force Test and Evaluation Command has identified certain characteristics which improve software maintainability. This information should be made available to the JLC for its consideration and potential application to the software development practices standard.

SOFTWARE ACQUISITION/DEVELOPMENT STANDARDS

PANEL A

CHAIRMAN: MR. JACK MUNSON

(213) 829-7511 (X-2787)

Systems Development Corporation  
ATTN: MR. JACK MUNSON  
2500 Colorado Avenue  
Santa Monica, CA 90406

COCHAIRMAN: MR. NORMAN BERMAN

(201) 544-2506

U.S. General Accounting Office  
ATTN: MR. NORMAN BERMAN  
11th Floor  
434 Walnut Street  
Philadelphia, PA 19106

SOFTWARE ACQUISITION/DEVELOPMENT STANDARDS

PANEL A

ARMY MEMBERS

Commander U.S. Army Electronic Proving Ground ATTN: STEEP-CS (MR. GRADY H. BANISTER, JR.) Fort Huachuca, AZ 85613	(602) 538-6068 AV 879-6068
Commander U.S. Army Missile Research and Development Command ATTN: DRDMI-TGG (MR. JERRY BROOKSHIRE) Building 4381 Redstone Arsenal, AL 35809	(205) 876-3366 AV 746-3366
Commander U.S. Army Communications Research and Development Command ATTN: DRDCO-TCS-BA-1 (MR. DAVID B. USECHAK) Fort Monmouth, NJ 07703	(201) 544-4011 AV 995-4011

SOFTWARE ACQUISITION/DEVELOPMENT STANDARDS

PANEL A

NAVY MEMBERS

Commander (703) 663-7311  
Naval Surface Weapons Center AV 249-7311  
ATTN: E33  
(MR. ROBERT CROWDER)  
Dahlgren, VA 22448

Commander  
Naval Weapons Center  
ATTN: Code 3108  
(MR. DENNIS FARRELL)  
China Lake, CA 93555

Commander (202) 692-2591  
Naval Sea Systems Command  
Code 06D  
(CAPT. JAMES E. RADJA)  
Washington, D. C. 20362

Commandant AV 224-4522  
Marine Corps  
ATTN: Code CCA-50  
(LTC. J. G. SCHAMBER)  
Room 3203  
Federal Building No. 2  
Washington, D. C. 20380

SOFTWARE ACQUISITION/DEVELOPMENT STANDARDS

PANEL A

AIR FORCE SYSTEMS COMMAND MEMBERS

LTC. CHARLES D. ADAMS  
ATTN: HQ AFCMD/EN  
Kirtland AFB, NM 87117

(505) 264-9626  
AV 964-9626

MR. ROBERT BERRI  
ATTN: SAMSO/AQT  
P.O. Box 92960  
Worldway Postal Center  
Los Angeles, CA 90009

(213) 643-1182  
AV 833-1182

MAJOR R. JOHNSON  
AFSC/XRF  
Andrews AFB, MD 20334

(301) 981-5731  
AV 858-5731

SOFTWARE ACQUISITION/DEVELOPMENT STANDARDS

PANEL A

AIR FORCE LOGISTICS COMMAND MEMBERS

MR. ROBERT ANDERSON (912) 926-5921  
ATTN: Warner Robins - ALC/MMECD AV 468-5921  
Warner Robins AFB, GA 31098

MR. DAVID THORNELL (801) 777-7231  
ATTN: Ogden ALC/MMECA AV 458-7231  
Hill AFB  
Ogden, UT 84056

MR. FRED WILSON (513) 255-6411  
ATTN: ASD/SD25L AV 785-6411  
Wright-Patterson AFB, OH 45433

SOFTWARE ACQUISITION/DEVELOPMENT STANDARDS

PANEL A

PANEL INVITEES

MR. JOSEPH P. CERAN  
Project Manager  
Computer Sciences Corporation  
Ten Route Thirty-five  
Red Bank, NJ 07701

(201) 747-6876

MR. GLENN GUSTAFSON  
Federal Systems Division, IBM  
Westlake Village, CA 91360

(805) 497-5151 (X-5432)

DR. THOMAS MARTIN  
RCA/GSD  
Building No. 206-1  
Cherry Hill, NJ 08358

(609) 338-5853

MR. M. G. MESECHER  
Manager  
Computer Systems Programming  
Sperry Systems Management  
Great Neck, NY 11020

(516) 574-1661

MR. READ MYERS  
Defense and Space Systems Group  
TRW Incorporated  
Mail Stop 55/5530  
One Space Park  
Redondo Beach, CA 90278

(213) 535-3446

MR. DON KANE  
MITRE Corporation  
Bedford, MA 01730

SOFTWARE ACQUISITION/DEVELOPMENT STANDARDS

PANEL A

PANEL INVITEES

MR. DON KANE  
MITRE Corporation  
Bedford, MA 01730

MR. ROBERT REINSTEDT  
RAND Corporation  
1700 Main Street  
Santa Monica, CA 90406

DR. ROBERT TAUSWORTHE  
Jet Propulsion Laboratory  
Stop 238-540  
4800 Oak Grove Drive  
Pasadena, CA 91103

## APPENDIX 2

### Bibliography for Software Acquisition/Development Standards Panel

#### 1. Copies reproduced and handed out to all panel members:

1. DOD Directive 5000.29	April 26, 1976	Management of Computer Resources in Major Defense Systems
2. AF Regulation 800-14, Vol. I and II	Sept. 26, 1975	Acquisition and Support Procedures for Computer Resources in Systems
3. Army Regulation 70-XX Draft	March 21, 1978	Management of Computer Resources in Army Defense Systems
4. SECNAV Instruction 5200.XX - Draft	-	Management of Computer Resources in Department of the Navy Systems
5. NAVELEX Instruction 5200.23	March 1, 1979	NAVELEX Computer Software Life-Cycle Management Guide
6. National Security Agency/Central Security Service Standard 81-2	Dec. 21, 1978	NSA/CSS Software Acquisition Manual
7. MIL-STD-1679 (Navy)	Dec. 1, 1978	Military Standard - Weapon System Software Development
8. Electronic Systems Division Draft	Feb. 27, 1979	Model, Statement of Work Task for Software Development
9. ASD TR-78-6	November 1977	Airborne Systems Software Acquisition Engineering Guidebook for Regulations, Specifications, and Standards
10. Vought Corporation ECS Software Document	Feb. 13, 1979	Government Embedded Computer Software Requirements and Related Activities
11. AIAA ECS Software Document	Feb. 20, 1979	STAMMP 6, Embedded Computer Software Documents

**II. Resources material used for reference:**

OMB Circular A-109	-	Major System Acquisition
OFPP Pamphlet No. 1	August 1976	Major System Acquisitions and Discussion of the Application of OMB Circular A-109
DODD 5000.1	Jan. 18, 1977	Acquisition of Major Defense Systems
DODD 5000.2	Jan. 18, 1977	The Decision Coordination Paper (DCP) and the Defense Systems Acquisition Review Council (DSARC)
DODD 5000.3	May 20, 1975	Test and Evaluation
DODD 5000.29	April 26, 1976	Management of Computer Resources in Major Defense Systems
DODD 5000.31	Nov. 24, 1976	Interim List of DOD-Approved Higher-Order Programming Languages (HOL)
DODI 5010.12	Dec. 5, 1968	Management of Technical Data
DODD 5010.19	July 17, 1968	Configuration Management
MIL-STD-482A	April 1, 1974	Configuration Status Accounting Data Elements and Related Features
MIL-STD-483	Dec. 31, 1970	Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs
MIL-STD-490	Oct. 30, 1968	Specification Practices
MIL-STD-499A	May 1, 1974	Engineering Management
MIL-STD-881A	April 25, 1975	WBS for Defense Material Items
MIL-S-52779	April 5, 1974	Software Quality Assurance Program Requirements

MIL-STD-1521A	June 1, 1976	Technical Reviews and Audits for Systems, Equipment and Computer Programs
MIL-Q-9858A	Dec. 16, 1963	Quality Assurance Requirements
MIL-S-83490	Oct. 30, 1968	Specifications, Types and Forms
AFR 57-1	Aug. 17, 1971	Policies, Responsibilities and Procedures for Obtaining New and Improved Operational Capabilities (ESD Supplement 10/30/73)
AFR 65-3	July 1, 1974	Configuration Management
AFR 80-14	Feb. 10, 1975	Test and Evaluation
AFR 300-1	June 10, 1971	Automatic Data Processing Program Management
AFR 300-2	Nov. 12, 1971	Management of Automatic Data Processing Systems
AFR 310-1	June 30, 1969	Management of Contractor Data
AFR 800-2	March 16, 1972	Program Management (AFSC Supplement 10/18/74 and ESD Supplement 2/25/74)
AFR 800-3	June 1, 1976	Engineering of Defense System
AFR 800-4	March 10, 1975	Transfer of Program Management Responsibility
AFR 800-6	July 14, 1972	Program Control—Financial (AFSC Supplement 9/4/74)
AFR 800-8	July 27, 1972	Integrated Logistics Support (ILS) Program for Systems and Engineering (AFSC Supplement 10/12/72)
AFR 800-9	April 25, 1973	Production Management in the Acquisition Life Cycle

<b>AFR 800-10</b>	<b>Sept. 12, 1973</b>	<b>Management of Multi-Service Systems, Programs and Projects</b>
<b>AFR 800-11</b>	<b>Aug. 8, 1973</b>	<b>Life-Cycle Costing (LCC)</b>
<b>AFR 800-12</b>	<b>May 20, 1974</b>	<b>Acquisition of Support Equipment</b>
<b>AFR 800-14 Volume I</b>	<b>May 10, 1974</b>	<b>Management of Computer Resources in Systems (AFSC Supplement 9/25/74)</b>
<b>AFR 800-14 Volume II</b>	<b>Sept. 26, 1975</b>	<b>Acquisition and Support of Computer Resources in Systems</b>
<b>SECNAVINST 3560.1</b>	<b>Aug. 8, 1974</b>	<b>Tactical Digital Systems Documentation Standards</b>

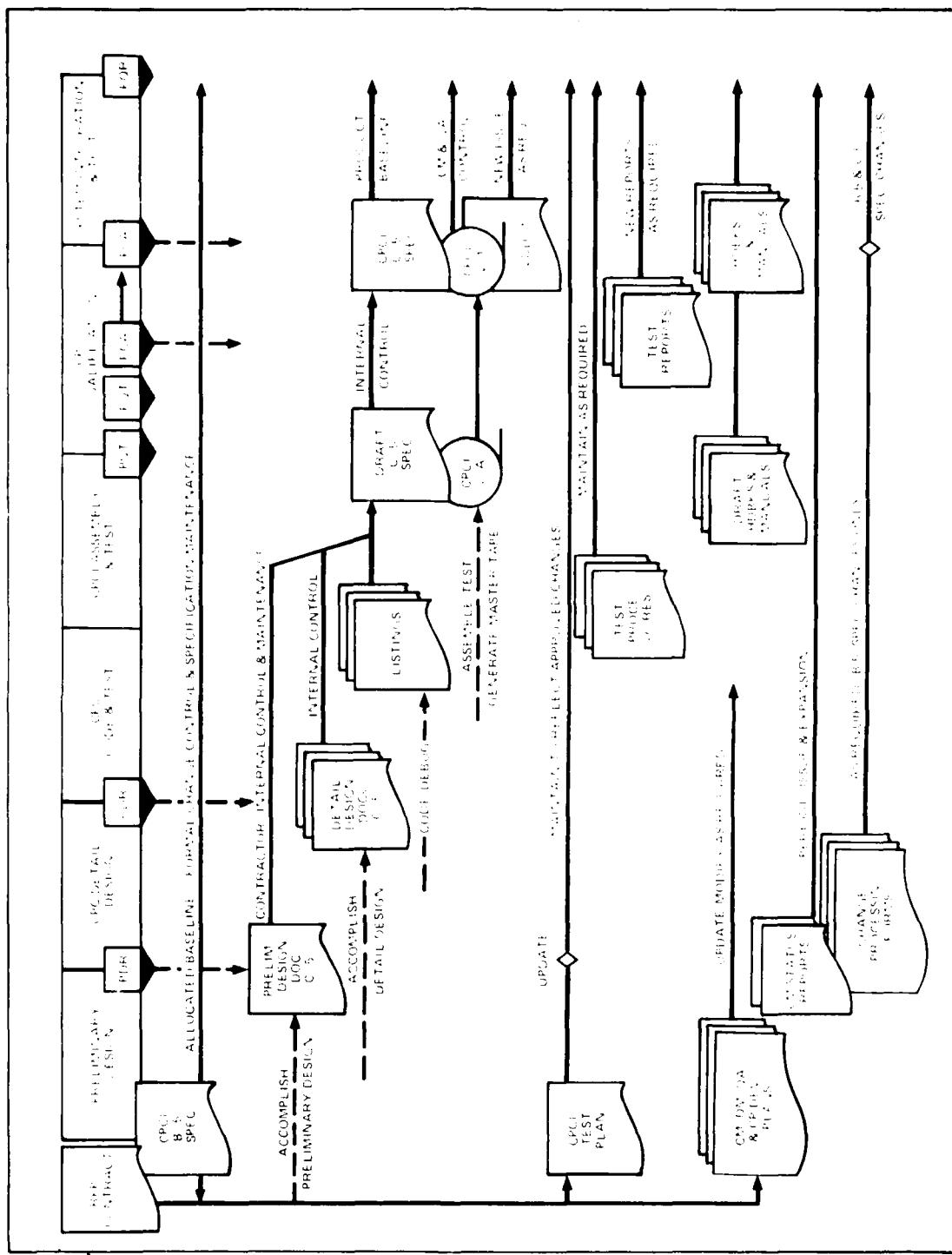
## APPENDIX 3 - STRAWMAN-EMBEDDED COMPUTER SYSTEM SOFTWARE DEVELOPMENT LIFE-CYCLE MODEL DESCRIPTION

The model portrayed in the accompanying figure is a proposed improvement to Figure 2-1 in AFR 800-14, Volume II. In addition, recommended text modifications to Chapter 2 of Volume II are outlined where appropriate.

One key assumption is made in the proposed model: the analysis effort required to derive the system processing requirements, and to allocate these to software and hardware requirements, was accomplished during the Demonstration and Validation phase of the Acquisition Life Cycle. Therefore, the Part I specification (as defined in Appendix VI of MIL-STD-483) is in close to final form when the Software Development Model starts.

The following commentary serves to explain the new model and to indicate needed changes to the text of AFR 800-14, should the proposed model be adopted.

<u>Para</u>	<u>Comment</u>
2-1	Satisfactory
2-2	Satisfactory
2-3	Satisfactory
2-4	Subparagraph (b) should indicate that the software development specifications are preliminary only in the sense of needing to reflect the design approach as necessary. The functional requirements should be complete.
2-5	Subparagraph (a) should reflect the nature of the material to be discussed at the PDR. Recent practice has evolved to using the Partial Draft of the Part II specification, as indicated in the figure. In addition, the Test Plan should be prepared to support the PDR, as indicated in the figure. It should also be noted that this subparagraph indicates that the effort going on is "preliminary design". Therefore, the "analysis" effort indicated in Figure 2-1 has been changed to Preliminary Design, consistent with the text. Finally, this subparagraph is an appropriate place to note the iterative nature of the entire development process, including Preliminary Design. Therefore, the feedback arrows of Figure 2-1 have been eliminated in favor of a discussion on the iterative nature of the process.  Subparagraph (b) should be modified to clarify the last sentence which vaguely alludes to incremental releases. The revised figure shows incremental releases more clearly and indicates that the overall product may be reviewed (incremental CDRs) as well as developed incrementally.  Subparagraph (c) does not differentiate clearly between the Test Plan and Test Procedures, but indicates the need for test information at CDR, with subsequent finalization prior to start of testing. This concept is portrayed in the new figure by the preparation of test procedures subsequent to CDR (the Test Plan meets the CDR need) and the finalization occurring at a new review called the Test Readiness Review. The TRR establishes that the procedures, test materials and test facilities are acceptable to initiate testing. This subparagraph also should couple the testing of the software to that of the system in which the software is embedded. This is indicated on the new figure.
2-6	Satisfactory



**Figure 3-1.** Embedded Computer System Software Acquisition Life-Cycle Model

- 2-7      Satisfactory
- 2-8      The introductory paragraph should be modified to reflect the new figure. The iterative nature of the process could be covered here instead of as proposed for 2-5a above.
- Subparagraph (b) should be replaced with a discussion of the elements of Preliminary Design. Some portions of this are in Subparagraph (b), so both of the subparagraphs must be revised to reflect the new figure. The need for authentication of the Part I development specification prior to PDR should be retained, but the rationale for this needed action should be provided in the text (i.e., the preliminary design cannot be completed if the functional requirements are not complete).
- Subparagraph (c) needs to be modernized to refer to more recent (and more acceptable) means of reflecting design than flow charts. Also, it is not clear that this phase addresses only the modules of the CPCI. The assembling of the modules into the overall CPCI is the next phase.
- Subparagraph (d) should cover the significance of the PQT. The PQT should be used to formalize the testing of critical modules of the system (but not all).
- Subparagraph (e), installation has been removed from the new figure, but the need for the ability to easily adapt the CPCI is required. This is a design requirement and should have been addressed in the development specification. The activity involved is part of the integration of the CPCI into the system.
- Subparagraph (f) should be expanded to note the user documentation shown on the revised figure.



AR 70-XX

MANAGEMENT OF COMPUTER RESOURCES IN

ARMY BATTLEFIELD AUTOMATED SYSTEMS

BY

DAVID USECHAK  
CENTER FOR TACTICAL COMPUTER SYSTEMS

CORADCOM



CENTACS



AR 70-XX

- IMPLEMENTS DOD DIRECTIVE 5000.29 - MANAGEMENT OF COMPUTER RESOURCES IN MAJOR DEFENSE SYSTEMS
- BASIC POLICY FOR LIFE CYCLE MANAGEMENT OF COMPUTER RESOURCES IN MAJOR AND DESIGNATED NON-MAJOR DEFENSE SYSTEMS
- OBJECTIVE IS TO ENSURE PROPER DEVELOPMENT OF COMPUTER RESOURCES IN ARMY DEFENSE SYSTEMS.



CENTACS



## COMPUTER RESOURCE MANAGEMENT PLAN (CRMP)

- IS DEVELOPED DURING DEMONSTRATION AND VALIDATION PHASE
- IS MAINTAINED THROUGHOUT SYSTEM LIFE CYCLE
- IS STRUCTURED TO INCLUDE QUALITY ASSURANCE AND CONFIGURATION MANAGEMENT
- IS USED AT ALL DIVISION LEVELS
- IS KEYED TO OVERALL SYSTEM ACQUISITION MILESTONES AND SCHEDULES
- IS STRUCTURED TO CONTAIN COMPLETE MANAGEMENT PLANNING INFORMATION E.G., ILS, CTP, QA,
- IS APPROVED BEFORE PROGRAM ENTERS FULL SCALE ENGINEERING DEVELOPMENT



CENTRAC S

CRMP OUTLINE



- THE CRMP HAS AS A MINIMUM THE FOLLOWING SECTIONS:

- A. GENERAL
- B. PROGRAM MANAGEMENT
- C. ACQUISITION MANAGEMENT
- D. DEVELOPMENT MANAGEMENT
- E. TEST AND EVALUATION
- F. POST DEPLOYMENT SUPPORT



CENTRAC S



### RESPONSIBILITY FOR CRMP

- THE MATERIAL DEVELOPER WILL ORCHESTRATE THE CRMP  
WITH THE COMBAT DEVELOPER, TESTERS, EVALUATORS  
AND POST DEPLOYMENT SUPPORT ACTIVITIES.



CENTACS



FUNCTIONS DEFINED BY AR 70-XX

- ESTABLISHES A SOFTWARE CONFIGURATION CONTROL BOARD (SCCB) AND COMPUTER RESOURCES WORKING GROUP (CRWG)
- ESTABLISHES DOD HIGHER ORDER PROGRAMMING LANGUAGES (HOLS) AND STANDARDS TO BE USED (DODI 5000.31)
- ESTABLISHES SPECIFIC RESPONSIBILITIES FOR DEPARTMENT OF ARMY STAFF AND MAJOR COMMANDS.



CENTRAC S



## KEY DOCUMENTS

- A. DODD 5000.29 - MANAGEMENT OF COMPUTER RESOURCES IN MAJOR DEFENSE SYSTEMS
- B. DODI 5000.31 - INTERIM LIST OF DOD APPROVED HIGH ORDER PROGRAMMING LANGUAGES (HOL)
- C. AR 18-1 - MANAGEMENT INFORMATION SYSTEMS: POLICIES, OBJECTIVES, PROCEDURES AND RESPONSIBILITIES
- D. AR 70-1 - ARMY RESEARCH, DEVELOPMENT, AND ACQUISITION
- E. AR 70-10 - TEST AND EVALUATION DURING DEVELOPMENT AND ACQUISITION OF MATERIEL
- F. AR 71-3 - USER TESTING
- G. AR 1000-1 - BASIC POLICIES FOR SYSTEM ACQUISITION
- H. AR 1000-2 -



CENTRAC S

**U.S. NAVY**

**COMPUTER SOFTWARE**

**DEVELOPMENT STANDARDS**

## **OVERVIEW**

- NEED FOR DEVELOPMENT STANDARDS
- DOD DIRECTIVE 5000.29 - MANAGEMENT OF COMPUTER RESOURCES IN MAJOR DEFENSE SYSTEMS
- MIL-STD-1679-WEAPON SYSTEM SOFTWARE DEVELOPMENT
- NAVELEX COMPUTER RESOURCES ACQUISITION MANAGEMENT PROGRAM
- CONCLUSIONS

**ROLE OF COMPUTER SOFTWARE IN TACTICAL  
SYSTEMS IS INCREASING - IT HAS BECOME  
CRITICAL TO SUCCESSFUL MAJOR WEAPON  
SYSTEM DEVELOPMENT**

- WEAPON SYSTEMS INVOLVING SOFTWARE -- 150
  - APPROX. 50% IN DEVELOPMENT
  - APPROX. 50% IN MAINTENANCE
- GREAT MAJORITY OF PROGRAM DEVELOPMENTS  
WITHIN NAVFLEX INVOLVE SOFTWARE

**ANNUAL WEAPON SYSTEM SOFTWARE COST  
WITHIN DOD IS ESTIMATED TO BE IN EXCESS OF  
\$1.5 BILLION**

- CONSERVATIVE ESTIMATE
- DIRECT COSTS ONLY
- EXCLUDES INTELLIGENCE, NONTACTICAL C<sup>3</sup>,  
LOGISTIC APPLICATIONS
- ESTIMATE 60% OF LIFE CYCLE SOFTWARE COST  
FOR MAINTENANCE

## WHAT HAS THE NAVY BEEN GETTING FOR ITS MONEY?

- RUNAWAY COSTS THAT DEFY MANAGEMENT CONTROL
- AN INABILITY TO MAINTAIN SCHEDULES
- THE DELIVERY OF SYSTEMS WHICH DO NOT MEET REQUIREMENTS
- DELIVERED SYSTEMS DIFFICULT AND COSTLY TO MAINTAIN

## WHAT ARE UNDERLYING CAUSES?

- INSUFFICIENT UNDERSTANDING OF SOFTWARE
- LACK OF QUALIFIED PERSONNEL
- LACK OF SOFTWARE LIFE CYCLE PLANNING
- INADEQUATE PROCUREMENT PACKAGES
- INADEQUATE SOFTWARE QUALITY CONTROL
- INADEQUATE CONFIGURATION MANAGEMENT
- LACK OF SOFTWARE MONITORING
- LACK OF MILESTONES, REVIEWS, AND AUDITS

DODD 5000.20

- COMPUTER RESOURCES LIFE CYCLE MANAGEMENT PLAN
- REQUIREMENTS VALIDATION
- CONFIGURATION MANAGEMENT OF COMPUTER RESOURCES
- MILESTONE DEFINITION
- GUIDANCE DOCUMENTS FOR PROGRAM MANAGERS
- ESTABLISHMENT OF TRAINING AND CAREER PATHS FOR TECHNICAL EXPERTS

## MIL-STD-1670

- PROVIDES REASONABLE STANDARDS FOR SOFTWARE DEVELOPMENT
- TIES DOCUMENTATION TO DESIGN AND DEVELOPMENT REQUIREMENTS
- DETERMINATION OF PERFORMANCE REQUIREMENTS
  - PERFORM ANALYSIS
  - DETERMINE FUNCTIONAL REQUIREMENTS
  - DEFINE SYSTEM RESOURCE REQUIREMENTS
  - RELATED DOCUMENTATION-PFS, IDS
- DEVELOPMENT OF PROGRAM DESIGN
  - BASED ON PERFORMANCE REQUIREMENTS
  - DESIGN ANALYSIS
  - TOP DOWN
  - RESOURCE RESERVES - 20 PERCENT
  - RELATED DOCUMENTATION – PFS, IDS

# MIL-STD-1673 (CONT'D)

## ● PROGRAMMING STANDARDS

- STRUCTURED PROGRAMMING
- SINGLE ENTRY/EXIT
- TRACEABILITY
- SIZE-AVERAGE 100 LINES, MAXIMUM 200 LINES
- LIMITED USE OF GOTO
- RELOCATABLE CODE
- RELATED DOCUMENTATION-PDD,PPD

## ● PROGRAMMING CONVENTIONS

- SYMBOLIC PARAMETERIZATION
- NAMING
- NUMERICAL CONSTANTS AND VARIABLES
- NARRATIVE DESCRIPTION
- FLOW CHARTS NOT REQUIRED
- RELATED DOCUMENTATION-PDD, PPD (STR)

## MIL-STD-1873 (CONT'D)

- PROGRAM PRODUCTION
  - TOP DOWN IMPLEMENTATION
  - INCREMENTAL DEVELOPMENT
  - APPROVED HOL
  - RESOURCE MANAGEMENT
  - UTILIZATION OF PROGRAMMING SYSTEM LIBRARY
  - LISTINGS
  - RELATED DOCUMENTATION - FDD, PPD
- PROGRAM REGENERATION
- PROGRAM OPERATION
- PROGRAM TESTING
  - LEVELS OF TEST
  - MODULE
  - SUBPROGRAM
  - PROGRAM PERFORMANCE (FORMAL)
  - SYSTEM(S) INTEGRATION
  - SOFTWARE TROUBLE REPORTING
    - ERROR PRIORITY
    - STR ACCOUNTING

## MIL-STD-1679 (CONT'D)

- PROGRAM ACCEPTANCE
  - ERROR LIMIT
  - PATCH LIMITS
  - TEST DURATION
  - STRESS TESTING
  - TEST ENVIRONMENT
  - SUPPORT PROGRAMS
  - REDUCED CAPABILITY TESTING
- SOFTWARE QUALITY ASSURANCE-SOFTWARE QA PLAN

## MIL-STD-1673 (CONT'D)

### ● SOFTWARE CONFIGURATION MANAGEMENT

- BASELINES
- CONFIGURATION IDENTIFICATION
- CONFIGURATION CONTROL
- CONFIGURATION ACCOUNTING
- SOFTWARE CHANGE PROPOSALS
- DOCUMENTATION CHANGES
- CONFIGURATION CONTROL BOARD
- RELATED DOCUMENTATION PLAN – SOFTWARE CM PLAN

### ● MANAGEMENT CONTROL

- MANAGEMENT ORGANIZATION
- RESOURCE REQUIREMENTS
- STATUS REVIEW
- INSPECTIONS AND AUDITS
- RELATED DOCUMENTATION-SOFTWARE DEVELOPMENT PLAN

## **COMPUTER RESOURCES QUALITY ASSURANCE PROGRAM**

- NAVFEX INSTRUCTION 5200.22 (DRAFT) IS  
CORNERSTONE OF PROGRAM
  - SETS POLICY FOR SOFTWARE DEVELOPMENT
  - SETS PRACTICES TO BE FOLLOWED IN ANY  
SOFTWARE DEVELOPMENT
  - PROVIDES FOR ESTABLISHMENT OF A COMPUTER  
RESOURCES ACQUISITION SUPPORT OFFICE  
WITHIN NAVFEX

COMPUTER RESOURCES, QA PROGRAM  
(CONTINUED)

- DEVELOPING GUIDEBOOKS FOR SOFTWARE LIFE CYCLE MANAGEMENT
- ESTABLISHING MEANS FOR ACCOUNTING AND DEVELOPING SOFTWARE ENGINEERS AND MANAGERS
- ESTABLISHING LIBRARY OF USER SOFTWARE

# **NAVELEX INSTRUCTION 5200.22**

## **• POLICY**

- THE BASIC RESPONSIBILITY FOR THE QUALITY OF COMPUTER RESOURCES ACQUIRED WITHIN THE NAVELEX ORGANIZATION RESIDES WITH THE COGNIZANT DIRECTORATE, PROJECT MANAGEMENT GROUP, OR FIELD ACTIVITY. EACH DEPUTY COMMANDER, PROJECT MANAGER, AND FIELD ACTIVITY COMMANDING OFFICER SHALL UTILIZE ESTABLISHED QUALITY ASSURANCE RESOURCES AND ADHERE TO THE PRACTICES SPECIFIED IN THIS INSTRUCTION TO ASSURE ADEQUATE QUALITY OF ALL COMPUTER SOFTWARE END PRODUCTS.**

NAVELEX INSTRUCTION 5200.22  
(CONTINUED)

• PRACTICES - FOLLOWED BY PROJECT MANAGERS  
FOR SOFTWARE DEVELOPMENT

- COMPUTER RESOURCES MANAGED AS SUBSYSTEM OF MAJOR IMPORTANCE
- COMPUTER RESOURCES LIFE CYCLE MANAGEMENT PLAN REQUIRED
- COMPUTER SOFTWARE DEVELOPED TO EXISTING STANDARDS
- SOFTWARE QA PLANNED AND BUDGETTED FOR
- VERIFICATION OF REQUIREMENTS REQUIRED BEFORE FSED
- EMPLOYMENT OF ADEQUATE CONTRACTURAL REQUIREMENTS FOR SOFTWARE
- ENFORCEMENT OF PRACTICE OF SOFTWARE QA BY DEVELOPING AGENCY LAW MIL-S-52779
- SOFTWARE CONFIGURATION MANAGEMENT LAW MIL-STD-48C
- IDENTIFICATION OF SPECIFIC MILESTONES TO ENSURE PROPER DEVELOPMENT

# **NAVELEX INSTRUCTION 5200.22**

## **(CONTINUED)**

### **● PRACTICES (CONTINUED)**

- SPECIFICATION OF ADEQUATE DOCUMENTATION TO ENSURE PROPER VERIFICATION AND MAINTAINABILITY**
- REQUIREMENT OF CONTRACTOR SOFTWARE DEVELOPMENT PLAN**
- PERFORMANCE OF ADEQUATE TESTING**
- PROVISIONS FOR ADEQUATE SOFTWARE EVALUATIONS, REVIEWS, AND AUDITS**

## NAVELEX INSTRUCTION 5200.22 (CONTINUED)

- ESTABLISHMENT OF NAVELEX COMPUTER RESOURCES ACQUISITION SUPPORT OFFICE (CRA30)
  - RECOMMEND SOFTWARE QA POLICY AND PROCEDURES TO COMMANDER NAVELEX
    - INTERPRET HIGHER AUTHORITY DIRECTIVES
    - RECOMMEND POLICY
    - ISSUE DIRECTIVES ON APPROVED POLICY AND PROCEDURES
    - PROVIDE POINT OF CONTACT FOR NAVELEX
  - REVIEW COMPUTER SOFTWARE ACQUISITION DOCUMENTATION
    - PROCUREMENT PACKAGES
    - SOFTWARE DEVELOPMENT TECHNICAL, PLANNING, AND MANAGEMENT DOCUMENTS

# NAVELEX INSTRUCTION E2000.22

## (CONTINUED)

### ● ESTABLISHMENT OF CRASSO (Continued)

- POLICE DEVELOPMENTS INVOLVING SOFTWARE FOR CONFORMANCE TO THIS INSTRUCTIONS
  - REVIEW OF ON GOING PROGRAMS
  - DETERMINE WHO COMPLY, WHO DO NOT COMPLY, AND DEGREE OF NON-COMPLIANCE
  - QUARTERLY REPORT TO COMMANDER NAVELEX
- PROVIDE CONSULTATION SERVICES TO ACQUISITION MANAGERS
  - GUIDANCE
  - REVIEW AND AUDIT
  - RECOMMENDATIONS ON PROBLEM AREAS
- PROVIDE SOFTWARE ENGINEERING/MANAGEMENT TRAINING TO NAVELEX PERSONNEL
  - ORIENTATION PROGRAMS
  - IN-DEPTH PROGRAMS TO DEVELOP SPECIALISTS

## **SOFTWARE MANAGEMENT GUIDES**

- o SOFTWARE LIFE CYCLE MANAGEMENT GUIDE  
(NAVELEX INST 5200.23)
- PART I - LIFE CYCLE ACQUISITION
  - o PHASES
  - o ACTIVITIES
  - o REVIEWS
  - o PLANNING

- PART II - SOFTWARE DEVELOPMENT AND MONITORING

- o TYPES OF SOFTWARE
- o SOFTWARE DEVELOPMENT
- o SOFTWARE MONITORING
- o DOCUMENTATION!
- o TESTING
- o QUALITY ASSURANCE
- o CONFIGURATION MANAGEMENT

# **SOFTWARE MANAGEMENT GUIDES**

## **(CONTINUED)**

### **- APPENDICES**

- GLOSSARY
- REFERENCES
- STATEMENT OF WORK
- CRDL
- SPECIFICATION
- OTHER RFP ITEMS - EVALUATION CRITERIA
- COMPUTER RESOURCE LIFE CYCLE MANAGEMENT PLAN

# **SOFTWARE MANAGEMENT GUIDES (CONTINUED)**

- ADDITIONAL SOFTWARE MANAGEMENT GUIDEBOOKS
- SOFTWARE QUALITY ASSURANCE AND DEVELOPMENT MONITORING
  - SOFTWARE QA
  - REVIEWS AND AUDITS
  - VERIFICATION AND VALIDATION
  - SOFTWARE DEVELOPMENT MONITORING
- CONFIGURATION MANAGEMENT
- COMPUTER SOFTWARE PLANNING, CONTRACTING, AND PROCUREMENT
- SYNOPSIS OF DIRECTIVES, STANDARDS AND SPECIFICATIONS
  - PLANNING AND PROCUREMENT
  - CONTRACTUAL REQUIREMENTS
- SERIES OVERVIEW

## **CONCLUSIONS**

- DEVELOPMENT STANDARDS ARE NEEDED
- DEVELOPMENT STANDARDS ARE HERE TO STAY

**Specifying Milestones  
for  
Software Acquisitions**

**R. E. Berri**  
**The Aerospace Corporation**

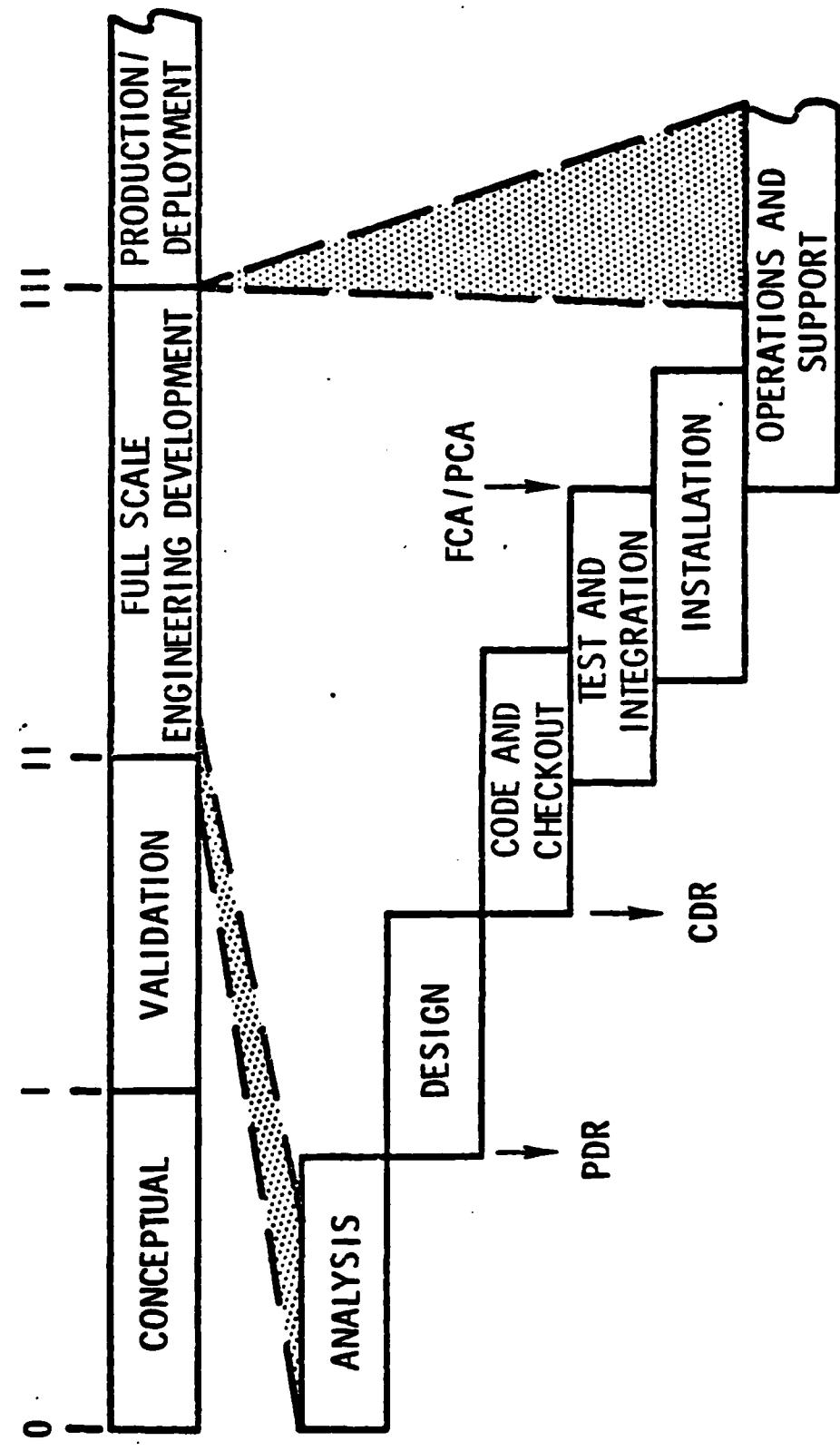
## **Outline**

- THE FRAME OF REFERENCE
- MANAGEMENT SYSTEMS COMPARISONS
- ADDITIONAL TECHNICAL CONSIDERATIONS
- RELATED CONTRACTUAL IMPACTS
- FUTURE ACTIONS AND SUMMARY

## A Frame of Reference

- DEPARTMENT OF DEFENSE POLICY
  - DOD DIRECTIVE 5000.29
  - EMBEDDED DATA PROCESSING SYSTEMS
  
- AIR FORCE ORIENTATION
  - AF REGULATION 800-14
  - ASSOCIATED MILITARY STANDARDS
  
- FULL SCALE ENGINEERING DEVELOPMENT
  
- DOCUMENTATION/MILESTONE MANAGEMENT  
RELATIONSHIPS

## Life Cycle Phases and Development Activities

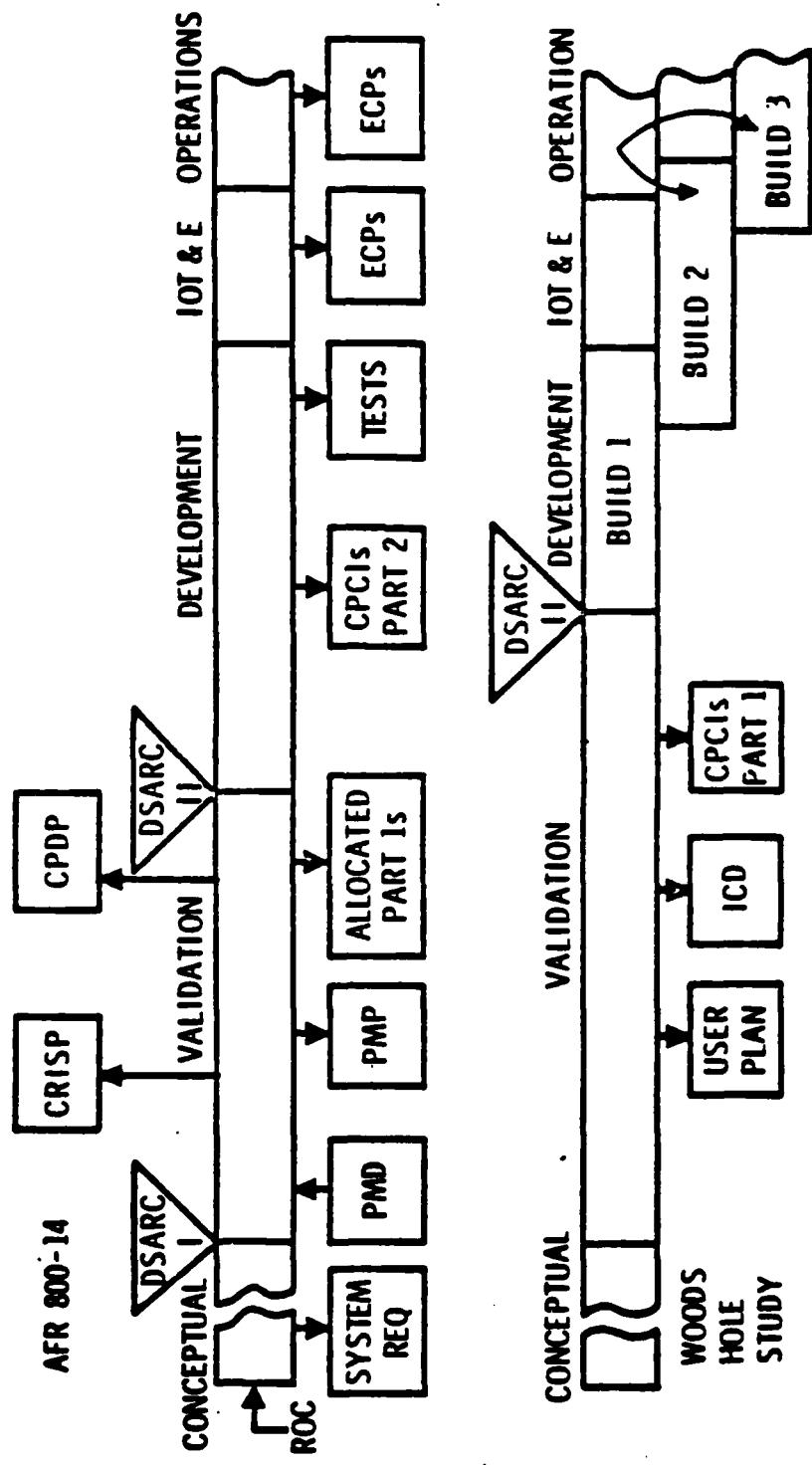


**PROBLEMS WITH AIR FORCE POLICY FRAME OF REFERENCE**

- POLICY LEVEL BELOW AFR 800-14 NOT COALESCED INTO MANAGEMENT SYSTEM
- INFORMAL POLICY (GUIDEBOOKS) DOES NOT ADDRESS ALL ACQUISITION SCENARIOS
- POLICY BASED ON OUTDATED MILITARY STANDARDS (MIL-STD-483, 490)
- PREVIOUS IMPROVEMENT EFFORTS FAIL TO GAIN MOMENTUM

# Woods Hole Study

## Impact on System Acquisition Cycle



## GUIDEBOOK LIST/DELIVERY

	AV	C3	AIE
REGULATIONS, SPECS, STDS	1	4	2
QUALITY ASSURANCE	1	1	3
REVIEWS & AUDITS	1	2	4
CONTRACTING	2	4	1
SOW'S & RFQ'S	2	3	1
VERIFICATION	2	1	3
VALIDATION/CERTIFICATION	2	1	3
CONFIGURATION MANAGEMENT	3	2	4
MEASURING STATUS	3	4	2
MEASURING COSTS	3	3	1
REQUIREMENTS SPEC'S	3	3	1
DOCUMENTATION	4	4	2
MAINTENANCE	4	1	4
MANAGEMENT REPORTS	-	2	4

## Key Reference Documents

- SYSTEM SPECIFICATION (SYS SPEC)
- DEVELOPMENT SPECIFICATION (DEV SPEC)
- PRODUCT SPECIFICATION (PROD SPEC)
- INTERFACE CONTROL DOCUMENT (ICD)
- TEST PLAN (T PLAN)
- TEST PROCEDURES (T PROC)
- TEST REPORT (T RPT)
- USERS MANUAL (U/M)
- PROGRAMMING MANUAL (P/M)

## Related Key Events

- SYSTEM DESIGN REVIEW (SDR)
- PRELIMINARY DESIGN REVIEW (PDR)
- CRITICAL DESIGN REVIEW (CDR)
- PRELIMINARY QUALIFICATION TESTS (PQT)
- FORMAL QUALIFICATION TEST (FQT)
- FUNCTIONAL CONFIGURATION AUDIT (FCA)
- PHYSICAL CONFIGURATION AUDIT (PCA)
- FORMAL QUALIFICATION REVIEW (FQR)

# DOD 4120.17M Comparison

<u>REFERENCE</u>	<u>EQUIVALENT</u>
● SYS SPEC	FUNCTIONAL DESCRIPTION
● DEV SPEC	SYSTEM/SUBSYSTEM SPECIFICATION DATA REQUIREMENTS DOCUMENT
● PROD SPEC	PROGRAM SPECIFICATIONS DATA BASE SPECIFICATION
● ICD	—
● T PLAN } ● T PROC }	TEST AND IMPLEMENTATION PLAN
● T RPT	TEST ANALYSIS REPORT
● U/M	USERS MANUAL
● P/M	PROGRAMMING MAINTENANCE MANUAL
—	✓ COMPUTER OPERATION MANUAL

# WS8506 Comparison

<u>REFERENCE</u>	<u>EQUIVALENT</u>
● SYS SPEC	—
● DEV SPEC	COMPUTER PROGRAM PERFORMANCE SPECIFICATION
● PROD SPEC	COMPUTER PROGRAM DESIGN SPECIFICATION
	COMPUTER SUBPROGRAM DESIGN DOCUMENT
✓	COMMON DATA BASE DESIGN DOCUMENT
● ICD	—
● T PLAN	COMPUTER PROGRAM TEST PLAN
● T PROC	COMPUTER PROGRAM TEST PROCEDURES
● T RPT	—
● U/M	—
● P/M	—
—	✓ COMPUTER PROGRAM PACKAGE
—	✓ COMPUTER PROGRAM OPERATOR'S MANUAL

# SEC NAV INST 3560.1 Comparison

<u>REFERENCE</u>	<u>EQUIVALENT</u>
● SYS SPEC	TACTICAL OPERATIONAL REQUIREMENT SYSTEM OPERATIONAL SPECIFICATION SYSTEM OPERATIONAL DESIGN
● DEV SPEC	PROGRAM PERFORMANCE SPECIFICATION FUNCTION OPERATIONAL SPECIFICATION FUNCTION OPERATIONAL DESIGN
● PROD SPEC	PROGRAM DESIGN SPECIFICATION PROGRAM DESCRIPTION DOCUMENT DATA BASE DESIGN
● ICD	INTERFACE DESIGN SPECIFICATION
● T PLAN	TEST PLAN TEST SPECIFICATION
● T PROC ● T RPT	TEST PROCEDURES AND REPORTS
● U/M	✓ COMMAND AND STAFF MANUAL SYSTEM OPERATOR'S MANUAL
● P/M	PROGRAM DESIGN MANUAL ✓ OPERATOR'S MANUAL

## Exhibit 61-47B Comparison

<u>REFERENCE</u>	<u>MILESTONE EQUIVALENT</u>
● SYS SPEC	1 - COMPUTER PROGRAM DESIGN CRITERIA
● DEV SPEC	2 - IMPLEMENTATION CONCEPTS AND TEST PLAN
● PROD SPEC	4 - COMPUTER PROGRAM DESIGN SPECIFICATIONS
● ICD	3 - COMPUTER PROGRAM INTERFACE SPECIFICATION (PART OF M/S 2)
● T PLAN	—
● T PROC	(PART OF M/S-4) —
● T RPT	—
● U/M	—
● P/M	—
	✓ 5 - COMPUTER SUBPROGRAMS AND ASSOCIATED SUPPORTING DOCUMENTATION AND TEST CASES
	✓ 6 - COMPUTER PROGRAM SYSTEM, ACCEPTANCE SPECIFICATION
	✓ 7 - COMPUTER PROGRAM OPERATING INSTRUCTIONS
	✓ 8 - COMPUTER PROGRAM SUBSYSTEM

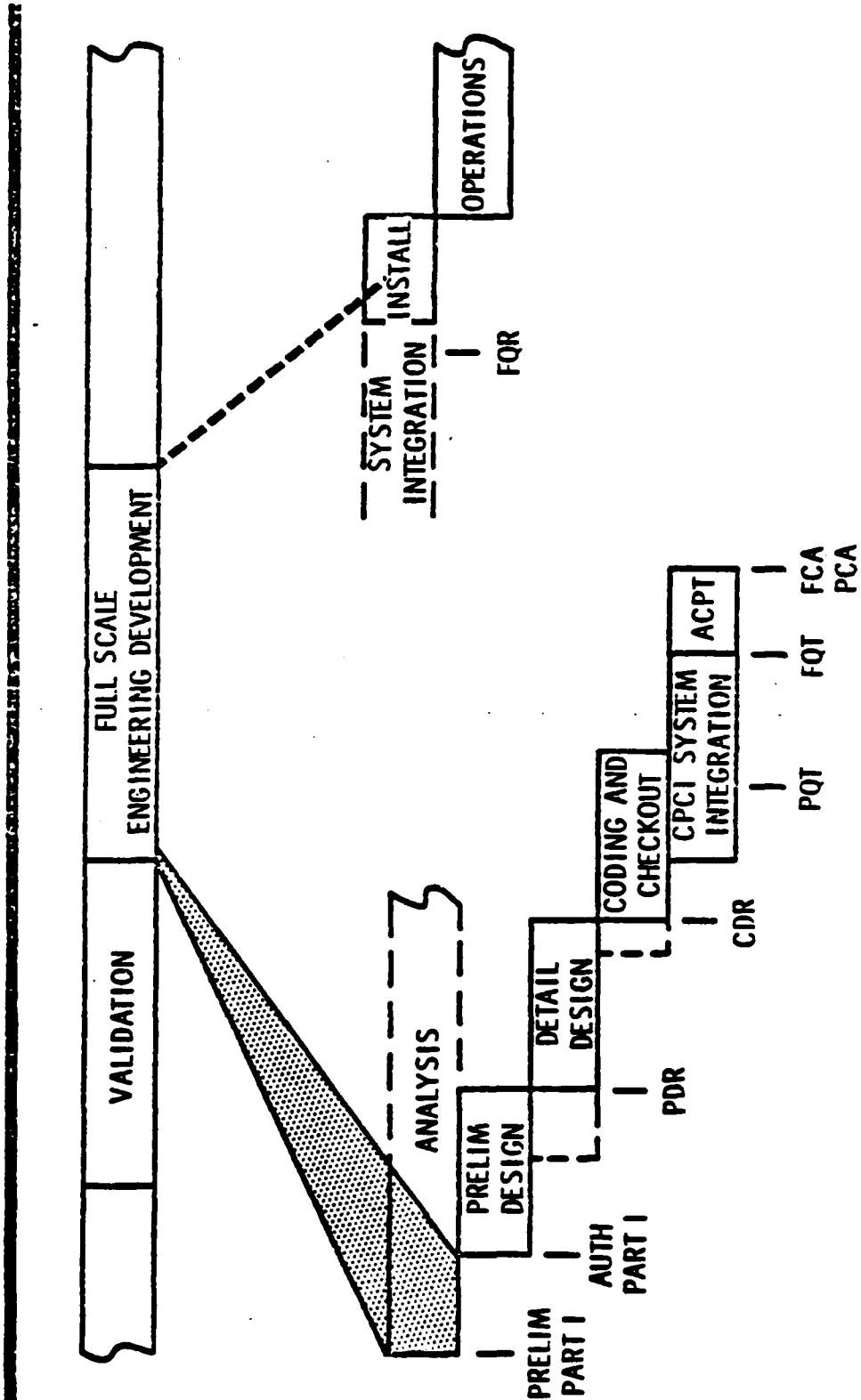
# Viking '75 Comparison

<u>REFERENCE</u>	<u>EQUIVALENT</u>
• SYS SPEC	INTEGRATED SOFTWARE FUNCTIONAL DESIGN SOFTWARE FUNCTIONAL DESCRIPTION
• DEV SPEC	FUNCTIONAL REQUIREMENTS DOCUMENT SOFTWARE REQUIREMENTS DOCUMENT
• PROD SPEC	GENERAL DESIGN DOCUMENT ✓ SOFTWARE DATA BASE DOCUMENT
• ICD	—
• T PLAN	TEST PLAN
T PROC	TEST PROCEDURES ✓ TEST DATA PACKAGE
• U/M	USER'S GUIDE
• P/M	—
—	✓ PROGRAM DESCRIPTION DOCUMENT

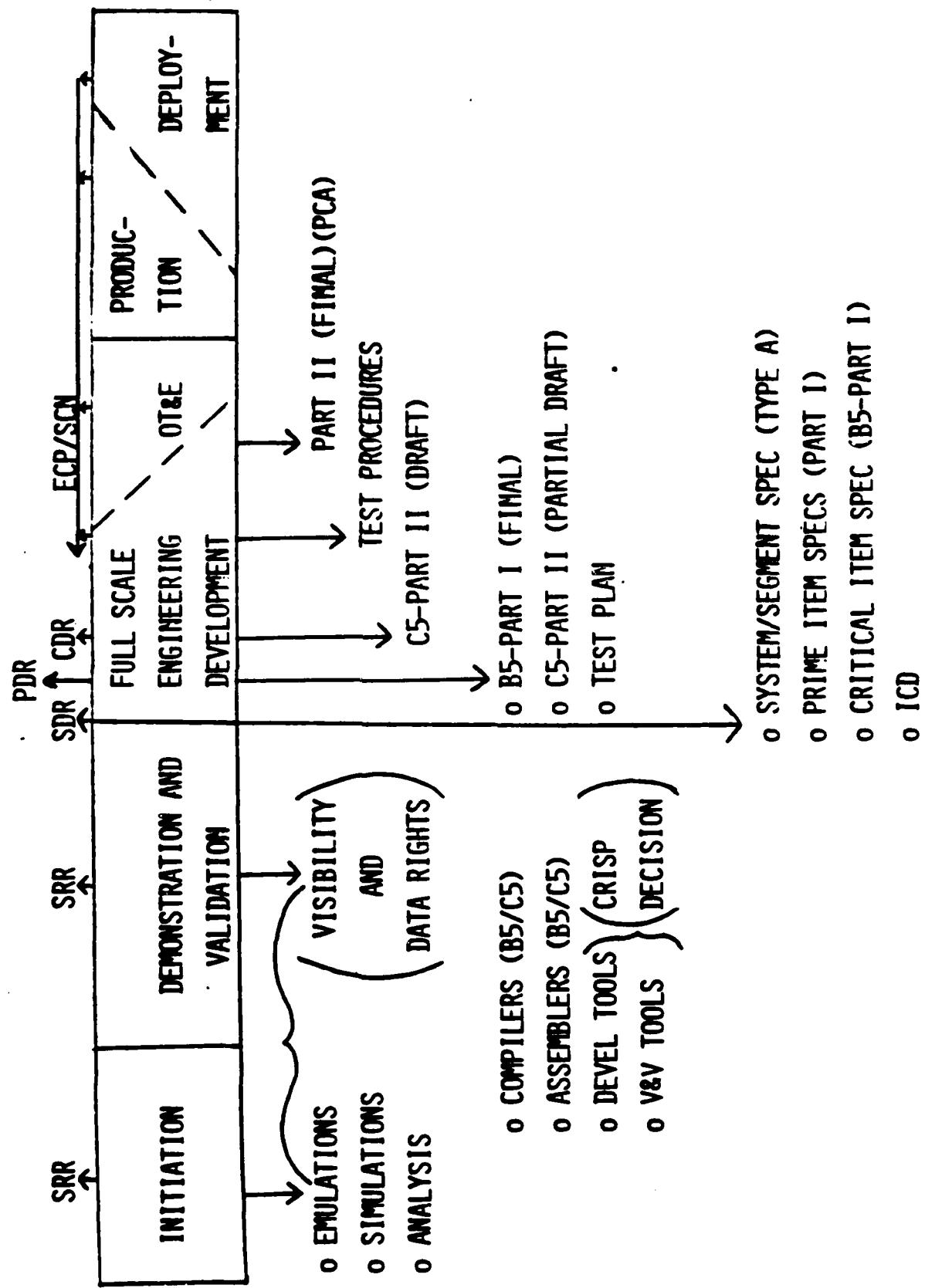
## **Composite Comparison Differences**

- DATA BASE SPECIFICATIONS
- COMPUTER OPERATION MANUAL
- COMPUTER PROGRAM PACKAGE
- INTEGRATION TEST PLANS/PROCEDURES
- INTEGRATED PROGRAM PACKAGE
- COMMAND AND STAFF MANUAL
- TEST DATA PACKAGE

## Revised Development Activities



## SOFTWARE SPECIFICATIONS IN THE LIFE CYCLE



## Technical Analyses To Establish Requirements

- FUNCTIONAL ALLOCATION AND FLOW
- HARDWARE/SOFTWARE/FIRMWARE TRADEOFFS
- SYSTEM/DATA SYSTEM INTERFACES
- DATA BASE CONCEPT
- PROCESSING LOAD (SIZING, TIMING, THROUGHPUT)
- LIMITS (ANOMALOUS AND EXTREME CONDITIONS)
- ACCURACY
- CRITICAL ALGORITHMS
- LANGUAGE SELECTION
- TESTABILITY

## **Management Analyses To Establish Requirements**

- COST/SCOPE COMPATIBILITY
- SCHEDULE
- NUMBER OF CONFIGURATION ITEMS
- PROGRAMMING STRUCTURE
- STANDARDS AND CONVENTIONS
- DEVELOPMENT FACILITIES
- GOVERNMENT FURNISHED MATERIAL
- MAINTENANCE METHOD
- SECURITY
- RELIABILITY AND QUALITY ASSURANCE

## **Additional Milestone Sources**

- DEVELOPMENT TOOLS
- DEVELOPMENT FACILITIES
- DESIGN REVIEW ACTIONS
- MODERN PROGRAMMING TECHNIQUES
- INDEPENDENT VERIFICATION AND VALIDATION
- INTEGRATION
- TECHNICAL INTERCHANGE MEETINGS
- SEPARATE TEST REVIEWS
- PRELIMINARY QUALIFICATION TESTS
- AUDITS
- INCREMENTAL DEVELOPMENT

## **Contractual Management Aids**

- COMPUTER PROGRAM DEVELOPMENT PLAN
- CONFIGURATION MANAGEMENT PLAN
- SYSTEM ENGINEERING MANAGEMENT PLAN
- QUALITY ASSURANCE PLAN
- TEST AND EVALUATION MASTER PLAN
- WORK BREAKDOWN STRUCTURE
- ENGINEERING CHANGE PROPOSALS

## **ADDITIONAL DISCUSSION ITEMS**

### **MODEL RELATED**

- ACQUISITION SCENARIOS
- LEVELS AND TYPES OF FCA/ PCA
- VERIFICATION/ VALIDATION/ CERTIFICATION
- TOOLS
- MAINTENANCE
  - / FACILITIES
  - / CPIN'S (OR EQUIVALENT)
- DATA ITEM CONSOLIDATION
- PRELIMINARY TESTING (PQT)
- STRUCTURED PROGRAMMING, BUILDS

## ADDITIONAL DISCUSSION ITEMS

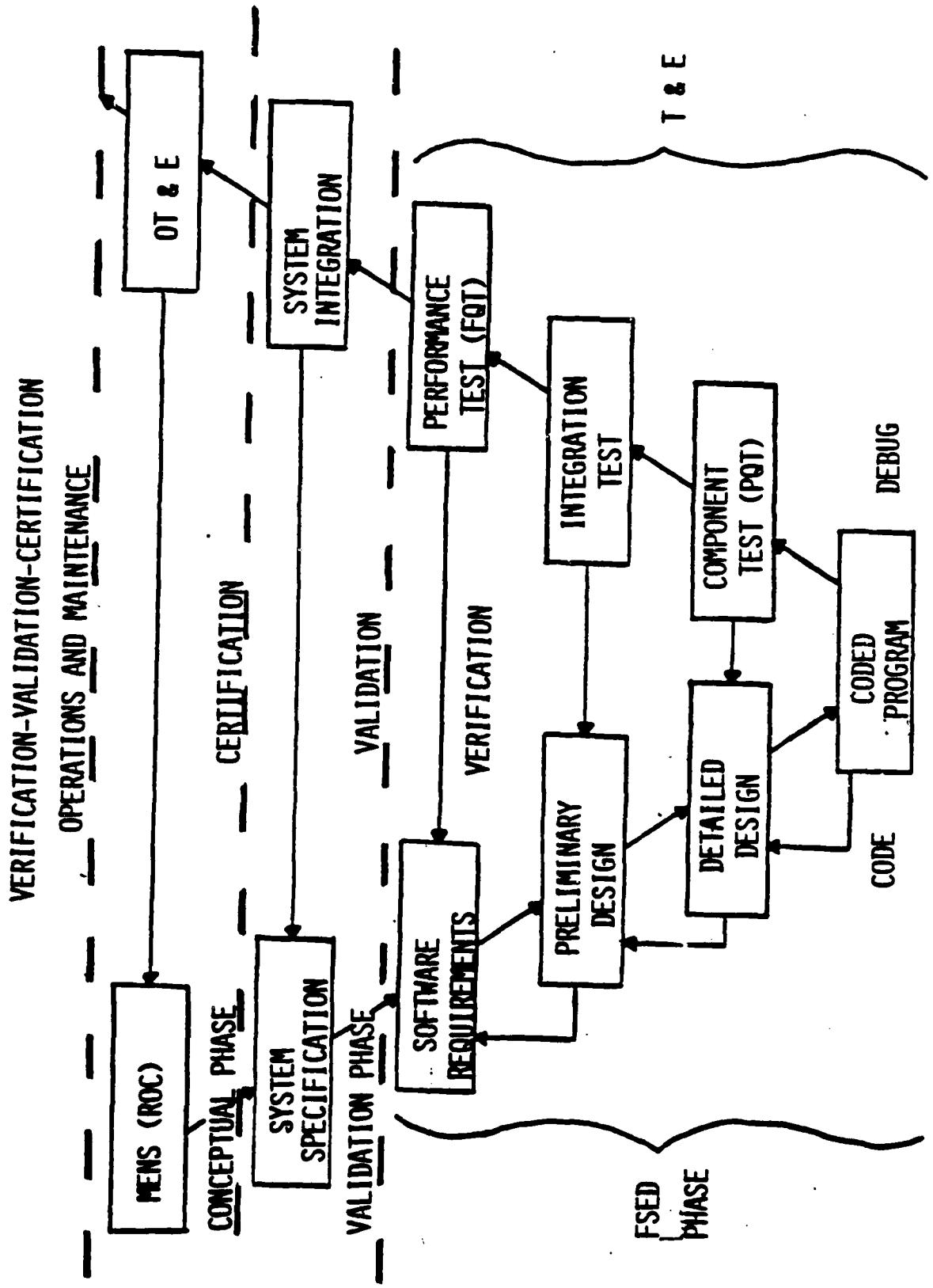
### SOMEWHAT RELATED

- TREATMENT OF FIRMWARE
- STANDARD DEFINITIONS
- SOFTWARE TREATMENT IN WORK BREAKDOWN STRUCTURE
- DATA RIGHTS TREATMENT
  - / CONTRACT SPECIAL PROVISIONS
  - / DATA ACCESSION LIST
  - / MINIMUM DOCUMENTATION STANDARDS
- NON-COMPLEX SPECIFICATION
- VALIDATION OF SOFTWARE MANUALS

PREFERRED DATA ITEMS



E30139 (E3119A)	PART I	A30567A	CPDP
E30140 (E3120A)	PART II	E3108	CMP
E30130	NON-COMPLEX	R30510	QAP
E3128	ECP	S30569	CRISD
E3134	SCN	-X- (E30141)	ICD
E3123	CHANGES	A6102/S3596A	SERD
T3703	TPL/TPR	M3410	UM
T3717	TR	M3411	PM
		-X-	STAND/CONV
		E3121	VDD



APPENDIX 5 - CHART INTEGRATING PDSS INTO ARMY ACQUISITION POLICY

AD-A103 485

AIR FORCE LOGISTICS COMMAND WRIGHT-PATTERSON AFB OH  
PROCEEDINGS OF THE JOINT LOGISTICS COMMANDERS JOINT POLICY COOR--ETC(U)

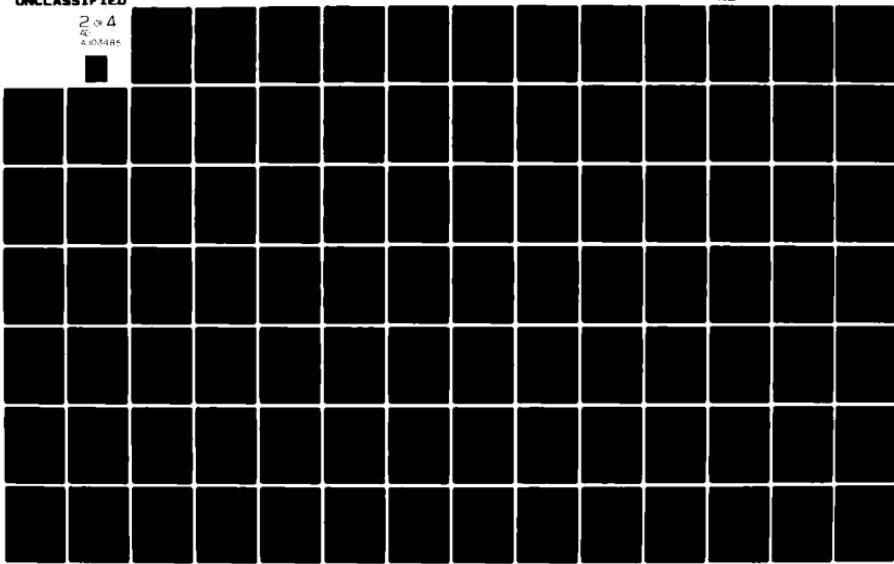
F/8 5/1

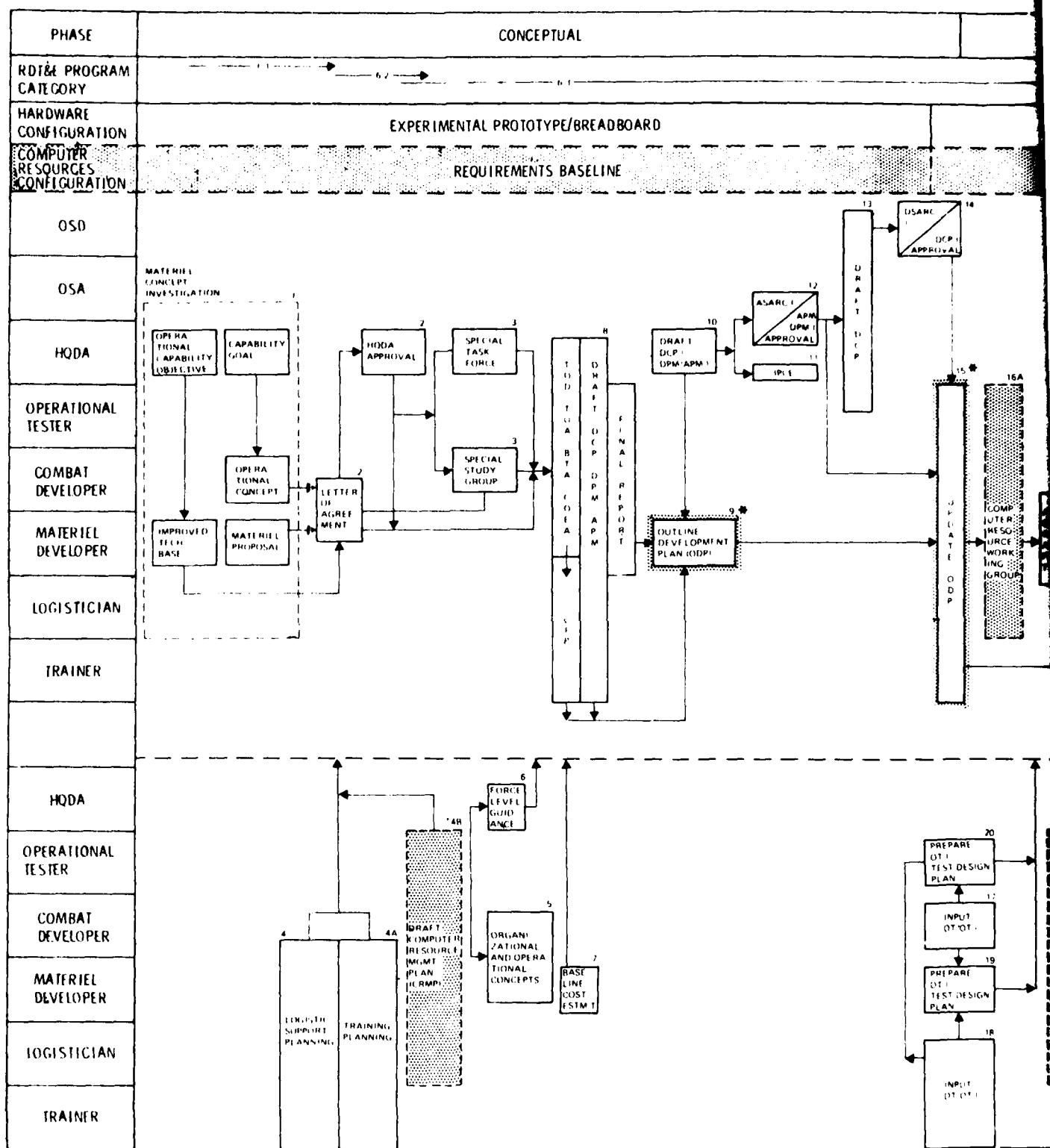
AUG 79

NL

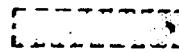
UNCLASSIFIED

2 or 4  
4c  
A103485

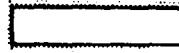




111111



461 VINT



REVISED EDITION

# LIFE CYCLE SYSTEM MANAGEMENT MODEL

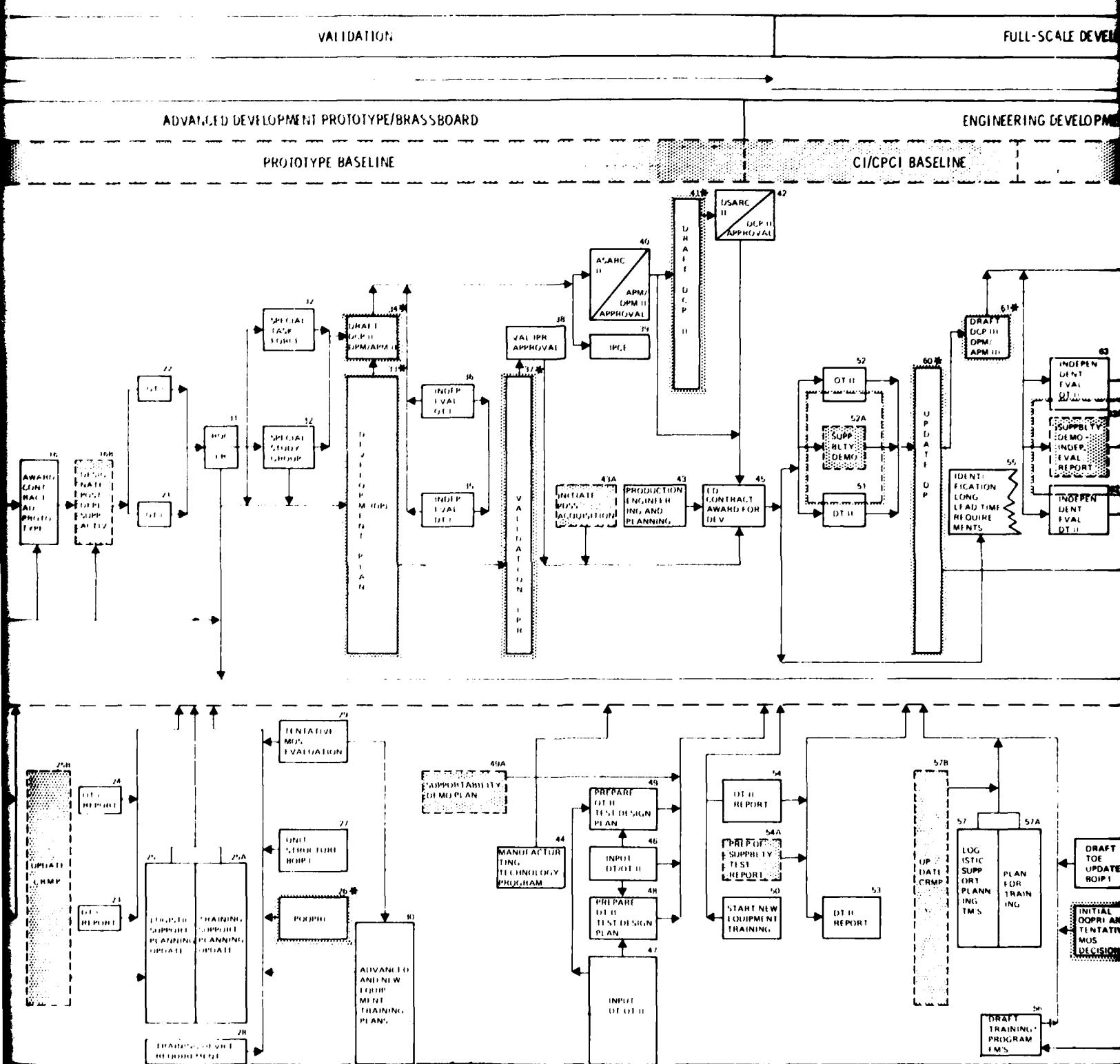
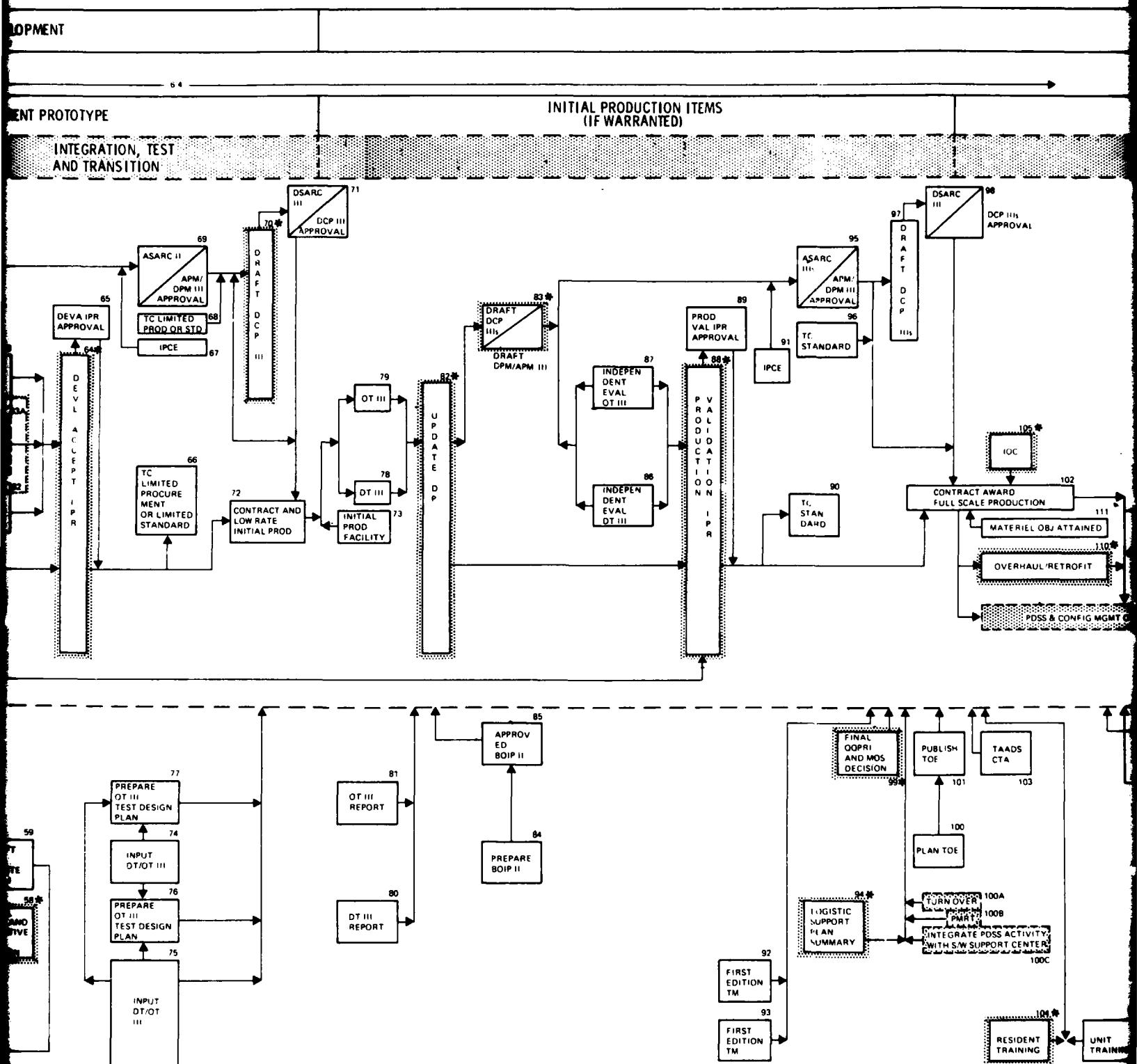
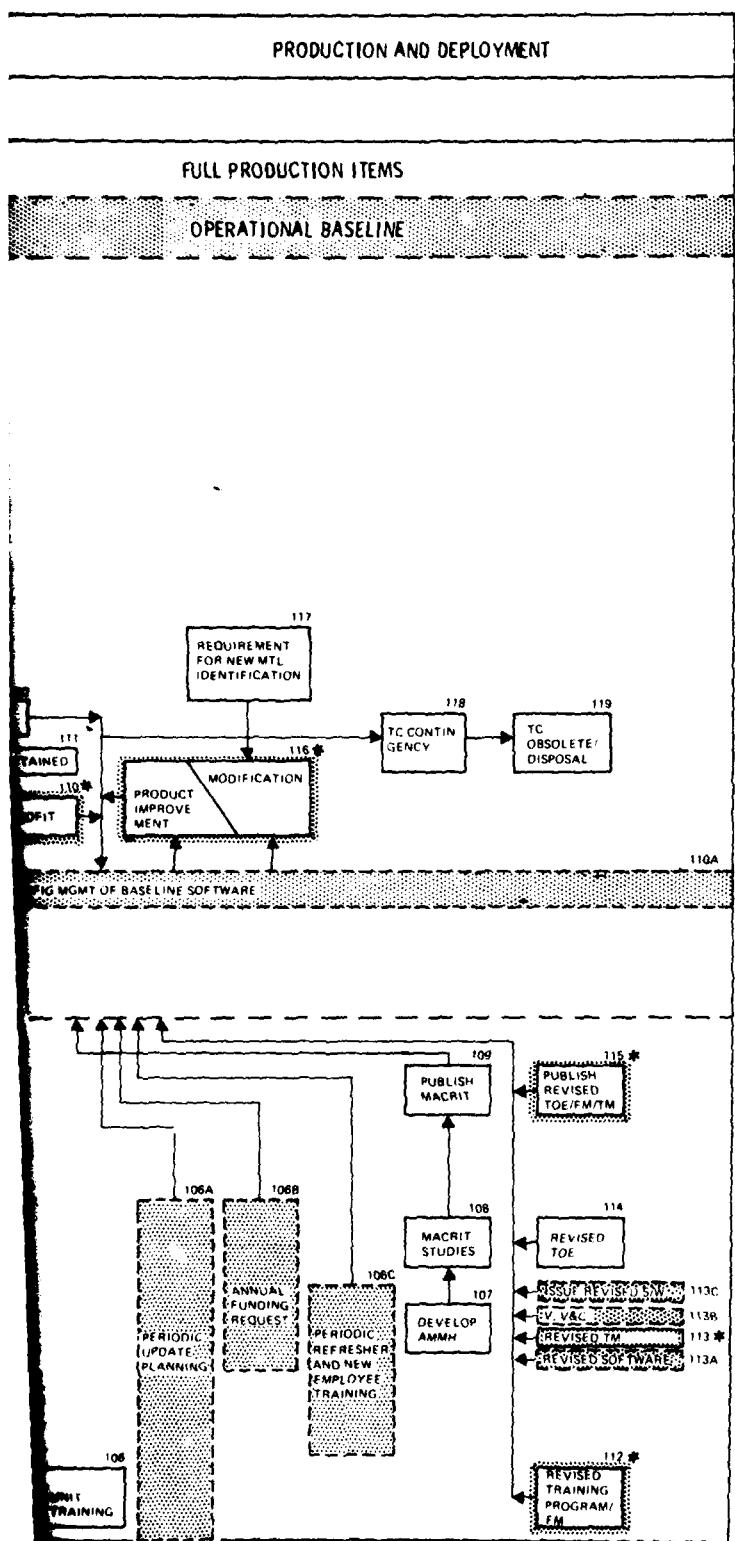


FIGURE C-1.





**PROCEEDINGS OF THE SOFTWARE WORKSHOP**  
**JOINT LOGISTICS COMMANDERS**  
**JOINT POLICY COORDINATING GROUP ON COMPUTER RESOURCE MANAGEMENT**  
**MONTEREY, CA, 2-5 APRIL 1979**

**Report of the Panel on**  
**SOFTWARE DOCUMENTATION**

**30 May 1979**

**Chairman:** Antonia D. Schuman  
TRW Inc.

**Co-Chairman:** Paul Shebalin  
NAVELEX

## OBJECTIVE

The assigned task of this panel was "to establish requirements for the minimization and standardization of computer software documentation as applied to the Department of Defense.

Considerations should include:

1. Purposes of documentation
2. Content and detail required
3. Evaluation of current documentation standards
4. Acceptance criteria for documentation
5. Long-range plan to generate DOD standard for software documentation."

The goal of the 3-day workshop in Monterey was to find a way to:

1. Reduce, combine, and standardize the Government's software documentation directives, regulations, standards, specifications, DIDs, and instructions that are imposed on industry
2. Make the software documentation produced by industry under Government contracts useful and useable. Eliminate or streamline the present massive and inconsistent set required.

## SCOPE

The problem, as expressed by the JLC-JPCG-CRM, was the proliferation of conflicting standards, specifications, instructions, and regulations for software documentation that have been issued by the different services. The systems for consideration were principally embedded rather than management information systems (MIS). The scope of our efforts included all documentation produced throughout a system's life cycle, starting with the MENS (ROC, SONS) and proceeding through development and test into the maintenance phase. The documentation discussed was not limited to those documents produced under contract, and included the CRISP (CRMP, CRLCMP) and nondeliverables. The panel decided not to consider the standardization of the CRISP, CRMP, and CRLCMP because they are Government-produced documents.

## APPROACH

The panel began its work by hearing presentations by each of the services on the documentation used. Considerable discussion accompanied these presentations, which highlighted many of the differences in terminology and requirements.

From these discussions, the panel decided that several issues should be addressed:

1. What documents are needed?
2. When, in the life cycle, are they needed?
3. How can common rules be enforced?

A tentative list of documents was generated by the entire panel, which then broke into four subpanels to consider which documents are needed and when:

1. Development
2. Test
3. Maintenance
4. Validation and Verification.

Each subpanel was charged with answering six questions about each document and, if time permitted, writing a description of the document. The questions were:

1. Who needs the document?
2. What purpose does it serve?
3. What is the relationship of the document to other documents?
4. Content and format?
5. Duration?
6. Applicability?

Finally, the entire panel met to produce the recommendations and determine the future course of action.

#### DISCUSSION

The issues raised by panel members during the open discussion included the following (in no particular order):

1. Definitions and terminology must be standardized.
2. There should be a common set of standards for software documentation, including MIS, C<sup>2</sup>, embedded, training, and ATE. The matter of acquisition regulations should not be a factor in the format, content, or terminology used.
3. Systems bought under warranty contracts must have delivered documentation.

4. Firmware is to be treated and documented as software.
5. We need a standard set of documents to describe the change process (trouble reports, system enhancements).
6. The ultimate maintainer of the software must have a useful set of documentation.
7. We need fewer, more complete documents.
8. What is CPCI, how are CPCIs determined, is it realistic to develop documentation based on CPCI?
9. What should be the scope of the application of the recommended standards? The panel was unable to decide whether they apply only to tactical systems or should apply to all DOD software, regardless of its application.
10. When in the life cycle are various documents needed? This issue was unresolvable, in part due to the lack of complete definition of the contents of each document and the belief of the panel that the schedule or documentation completion was more properly within the scope of the Acquisition Panel.

The serious issue of enforcement became a major topic of discussion during the entire workshop. The basic premise of MIL-STD-490 was held to be valid. However, 490, even as modified by 483, only covers specification and omits such things as user manuals.

It was felt by all participants that the best parts of each of the present specs be combined when the final MIL-Spec (or MIL-H) is written. The most important of these are:

1. DOD Standard 7935.1, "Automated Data Systems Documentation Standards"
2. MIL-STD-490
3. MIL-STD-483 (USAF)
4. MIL-STD-1679 (Navy)
5. MIL-STD-1521 (USAF).

The total set of references is presented in Appendix 2. Finally, the plethora of DIDs must be reduced and combined into a simple, uniform set.

The documents proposed by the panel are meant to constitute the complete set of software-related documents for tactical embedded computer systems within DOD. Individual project managers will be free to use subsets as appropriate for their projects, but the additional software-related documents as part of the RFP should be forbidden. The issue of whether the master list should be defined in a MIL-S or a MIL-H was one that divided the panel, with the key point being the modifiability of the controlling document.

## RECOMMENDATIONS

The panel recommended that the existence of other committees dealing with the same problem be determined and our efforts be coordinated with theirs. In addition, the work of the Acquisition Panel must be continued in order to place the set of recommended documents in the software life cycle.

A set of uniform DOD software documentation standards should be developed in four steps as follows:

1. Identify all documents needed for software development.
2. Modify MIL-STD-490 as follows:
  - a. Move A, B5, C5 specification instructions from MIL-STD-483 to MIL-STD-490
  - b. Enhance existing descriptions.
3. Create new MIL-STD to tie all documents together; tie to 490/483 and identify format and contents of those documents not in 490/483.
4. Attack DID problem. Come up with one set for documents listed in 1. above.

At a subsequent meeting of this panel on 2, 3 May 1979, their recommendations were as follows:

1. The list of documents (Appendix 3) needed for software development is intended to be used as a recommended list for which items could not be substituted.
2. The form of the document description, format, and definition should be contained in two MIL-STDs (e.g., 490 and the new standard) and supplemented by DIDs.
3. These two MIL-STDs are intended to be documentation standards and act as the mechanism for requiring the use of standard documents such as procuring agencies.
4. The mechanism for requiring use of the MIL-STDs is the contract CDRL.
5. The mechanism for superseding existing standards and descriptions is new DIDs.

Using the descriptions of each proposed document (Appendix 3), the panel recommended that the JLC fund a task to complete Steps 2, 3, and 4 of our original recommendation. In addition, it recommended that the task:

1. Include a review of MIL-STD 1679 (Navy) and MIL-S-52779 in order to identify the changes required to make them compatible with our list of documents

2. Make necessary revisions to MIL-STD 1521 so that it includes the documents to be reviewed and how the review should be performed
3. Further identify potential problems caused by the format contained in MIL-STD 490.

It is further recommended that this task be accomplished through a contract with a private contractor (excluding those contractors with members on the panel) and that a funded steering committee made up of members of this panel be appointed to write the SOW, monitor task progress, and review the results. The contract should be awarded around January 1980 with a period of performance of 6 months and a ceiling cost of \$99,000.

Finally, the JLC should begin laying the ground work to ensure DOD-wide acceptance of the new standards.

APPENDIX 1 - PARTICIPANTS

## APPENDIX 1 - PARTICIPANTS

Chairman: Antonia D. Schuman, TRW

Co-Chairman: Paul Shebalin, NAVELEX

### Subpanels

#### 1. Development

Chairman: Russell Eyres, NAVMAT  
Lloyd Searle, Planning Analysis Research Institute  
Gene Sydow, Litton DSD  
Grace Dugas, ESD (USAF)  
Marlene Hazle, MITRE Corporation  
Newnam Thompson, System Development Corporation

#### 2. Test

Chairman: Pat Ward, USATECOM (Army)  
George Weekly, USAF  
Kurt Fischer, Computer Sciences Corporation  
David Lee, NAVMAT

#### 3. Maintenance

Chairman: Verlon Duncan, Ogden ALC (USAF)  
Lee Stewart, MIRADCOM (Army)  
Don Wagus, Computer Systems Command (Army)  
Bob Sauer, Marine Corps Tactical Systems Supply Activity  
Harry Jennings, Warner Robins ALC (USAF)  
William Egan, Pacific Missile Test Center (Navy)

#### 4. Validation

Chairman: Paul Shebalin, NAVELEX  
Mike Landes, RADC (USAF)  
Gene Sievert, Teledyne Brown Engineering

APPENDIX 2 - SOFTWARE DOCUMENTATION REFERENCES

## APPENDIX 2 - SOFTWARE DOCUMENTATION REFERENCES

1. DODI 5010.12 (December 1968)  
"Management of Technical Data"
2. DOD Standard 7935.1-S (September 1977)  
"Automated Data Systems Documentation Standards"
3. MIL-STD-490 (May 1972)  
"Specification Practices"
4. MIL-STD-480 (October 1978)  
"Configuration Control - Engineering Changes, Derivations and Waivers"
5. MIL-STD-961 (March 1977)  
"Outline of Forms and Instructions for the Preparation of Specifications and Associated Documents"

### Air Force

6. AFR 800-14, Vol. II (September 1975)  
"Acquisition and Support Procedures for Computer Resources in Systems"
7. MIL-STD-483 (USAF) (June 1971)  
"Configuration Management Practices for Systems, Equipment and Computer Software"
8. MIL-STD-1521A (USAF) (June 1976)  
"Technical Reviews and Audits for Systems, Equipments and Computer Programs"
9. AFR 310-1 (June 1969)  
"Management of Contractor Data"
10. AFSC/AFLC Manual 375-7 (March 1971)  
Configuration Management for Systems, Equipment, Munitions, and Computer Programs"

### Navy

11. MIL-STD-1679 (Navy) (December 1978)  
"Weapon System Software Development"
12. SECNAVINST 3560.1 (August 1974)  
Tactical Digital System Documentation Standards"
13. NAVORD WS-8506 (November 1971)  
"Requirements for Digital Computer Program Documentation"

14. SECNAVINST 5233.1A, C1 (August 1974)  
"Department of the Navy Automated Data System Documentation Standards"

Army

15. USACSC Manual 18-100, C2 (October 1974)  
"Tactical Automatic Data Processing Systems Development, Maintenance and Documentation Standards"

DIDs

Air Force  
Navy

APPENDIX 3 - DESCRIPTIVE PARAGRAPHS

### APPENDIX 3 - DESCRIPTIVE PARAGRAPHS

(CRISP)

Interface Documents

Computer Program Development Plan

System Specification

Software Configuration Management Plan

Principles of Operation

Computer Program Requirements Specification

Preliminary Program Design Specification

Detailed Program Design Specification

Computer Program Product Specification

Version Description Document

Data Base Specification

Positional/Station Handbook

Computer Operator's Manual

S/W Maintenance/Programmer's Manual

Command Staff Manual

Diagnostics Manual

Software QA Plan

System/CPCI Test Plan

System/CPCI Test Specification

System/CPCI Test Procedures

System/CPCI Test Report

Trouble Report

## **INTERFACE DESIGN SPECIFICATION**

The purpose of this specification is to establish a set of requirements for the design determinations of the interdigital processor interfaces in the areas of tactical digital systems. It provides a detailed logical description of all data units, messages, buffer size, and use of all control signals for defining interdigital processor communication conventions such as enable/disable communications between the digital processors of the two specific systems. It also assigns control and responsibility for every control signal and data item passed between the interfacing systems.

## **INTERFACE CONTROL DOCUMENT**

It was proposed that a second document be prepared, called an Interface Control Document, which would be required regardless of the existence of currently implemented interfaces. This document would be the basis for determining if other systems should interoperate with the subject systems and for defining the requirements to achieve that interoperability. It would describe the functions and formats necessary for any future system to be able to interface as well as the hardware and software interface requirements to enable any future systems to be easily interfaced with the existing system.

## **COMPUTER PROGRAM DEVELOPMENT PLAN**

The Computer Program Development Plan (CPDP) is a document in which the contractor describes his plans for the management and development of the computer programs and associated documentation that he needs for the completion of the contract; this may include management of support software such as simulations and analysis tools, as well as operational programs and automatic test equipment programs. The CPDP addresses, summarizes, or references the management issues in a single document. The CPDP is started at RFP early in the life cycle, and is developed incrementally during the early stages of the software development cycle. Since the CPDP provides for the plans only, it is possible to complete it early. Because the final version of the CPDP is detailed to the lowest possible level, much of the information is not available when the document is initiated. At each stage of the life cycle, the document is fleshed out to the greatest extent possible as it becomes appropriate to fill in details. At 30 or 90 days after contract award (depending on the nature of the project), the CPDP should be complete. It is the document that provides a vehicle for outlining how the contractor will manage the software engineering elements of the requirements. The CPDP should be put on contract to serve as a base for assuring that the contractor does what he said. The DIDs describing the CPDP details are intended to be tailored for each application.

## SYSTEM SEGMENT SPECIFICATION

The system/segment specification serves two main purposes. First, it documents the functional and performance requirements defined for the system and/or segment as a result of studies of alternative system concepts. Second, after review and approval, it serves the configuration manager as the documentary tool to establish and control the functional baseline for the system and/or segment throughout the system acquisition cycle.

Throughout the acquisition process, up to the production phase, the system/segment specification provides the requirements for top-level testing. During the production and deployment phase and in post-deployment maintenance, the system/segment specification continues to be the controlled functional baseline, and is kept up-to-date to reflect approved system enhancements and modifications.

At the discretion of the program manager, the specifications may be divided into one volume containing general requirements common to all system segments, plus an additional volume for each system segment. In either case, the system will be defined by a single specification comprised of one or more volumes. This specification will contain the following elements:

1. Identification of the general system configuration in terms of system segments and/or functional areas
2. Definition of performance requirements and constraints for the system as a whole, and allocations of those to system segments and/or functional areas
3. Definitions of functional and physical interfaces between the system and external systems and between all functional areas within the system
4. Descriptions of the organizational and operational relationships relevant to the system and/or system segments
5. Descriptions of facilities, maintenance, personnel, and training concepts
6. Requirements for system/segment-level testing, including methods for verifying system and/or segment performance.

The system/segment specification will be expanded during the demonstration and validation phase to include lists of equipment and computer program configuration items, a specification tree, and definition of functional interfaces among system segments and/or functional areas.

## **SOFTWARE CONFIGURATION MANAGEMENT PLAN**

The Software Configuration Management Plan (SCMP) describes the contractor's internal computer software configuration management organization; the responsibilities of the members; the relationship among the several offices/divisions; the policies and procedures for identifying the documentation of the functional and physical characteristics of configuration items required by the contract; procedures for controlling changes to configuration items during the development; procedures for recording and reporting change-processing implementation status; and the external relationships required to maintain total system compatibility.

The SCMP provides the contractor the means to consolidate all policies, procedures, organizational descriptions, resources, and schedules relating to software configuration management in one document. The SCMP provides the procuring activity with detailed knowledge of the contractor's configuration management. Through the SCMP, the procuring activity can monitor the contractor's application of configuration management principles in conformance with standards invoked in the contract.

## **PRINCIPLES OF OPERATION DOCUMENT**

The purpose of the Principles of Operation Document is to provide explicit description of the way in which the data processing system will appear to its users and the way in which it will interact with them. It also describes assumptions about how the user will use the system to perform his operational tasks. It provides a vehicle for concurrence about the appearance and interaction of the system among the user, procuring agency, and developer. It is derived from: System Operational Concept, System Specification, CI and CPCI identification, knowledge of current operations and environment, etc. It provides guidance for development of hardware and software requirements and positional handbooks.

## COMPUTER PROGRAM REQUIREMENTS SPECIFICATION

The Computer Program Requirements Specification (CPRS) describes the operational and functional requirements of a computer program configuration item (CPCI) necessary for the design, test, and maintenance of the CPCI. It identifies the relationship of the CPCI to equipment and other CPCIs. It also identifies other constraints and standards in accordance with which the CPCI is to be designed and developed. The CPRS contains requirements rather than design and is reviewed before PDR. The CPRS is derived from the system/segment specification, principles of operation, hardware configuration and architecture, etc.; it is the primary source for the computer program product specification. Issues associated with the CPRS are the appropriate level of detail, the overlap with interface and data base specifications, and the differentiation between requirements and design.

## **PROGRAM DESIGN REVIEW DOCUMENT**

This is a performance-level specification that states functional performance requirements including program sizing, internal interfacing, data base constructs, linkage and sequencing of program parts, functional flow block diagrams, etc. This document contains specific program performance requirements for the total system or CPCI software, including inputs, processing, outputs, special requirements, and interfaces to intersystem functions/devices/processes not directly addressed by the system under development. The PDR document is the written vehicle which establishes performance targets and agreed upon system decomposition/partitioning which occurs during FSED, primarily at successful completion of PDR. The document is the outline for the PDR process and should contain:

1. Computer program functional flow
2. Storage allocation/timing allocation
3. Control function description
4. Program structure
5. Module identification and description
6. Security requirements and access controls
7. Development facility and tools to be used
8. Global data base architecture
9. Critical timing requirement
10. Functional traceability matrix.

## **CRITICAL DESIGN REVIEW DOCUMENT**

This is a design-level specification that states design of program(s), subprogram(s), module(s), and subroutine(s) in adequate detail to support coding of the respective modules. This document contains all of the performance objectives of the PDR document plus updated details of timing/sizing resource allocation, module development scheduling, and data formats. The CDR document is the "code to" design specification, which when approved at CDR will go into configuration control and serve as the documented guidance for implementation of programs, subprogram modules, etc. The document is the outline for the critical design review and should contain all PDR document subjects (by reference or inclusion) and the following:

- 1. Local data design**
- 2. Global data references**
- 3. Subprogram/module crops reference**
- 4. Detail logic description(s)**
- 5. Internal control structure**
- 6. Interaction conditions**
- 7. Register and index usage**
- 8. Calling sequences**
- 9. Traceability matrix**
- 10. Updated PDR data, if applicable.**

## **COMPUTER PROGRAM PRODUCT SPECIFICATION**

The Computer Program Product Specification documents the detailed technical description of the CPCI to be delivered under the terms of the contract.

The Computer Program Product Specification is derived from the Computer Program Development Specification and (if available) the Interface Design Specification. The Computer Program Product Specification is a baseline document, and is published at the end of the development phase. This specification shall specify the design description of the CPCI based upon the functional, performance, interface, and design requirements defined in the Computer Program Development Specification and the Interface Design Specification and shall contain all of the information in both the CDR and PDR documents by inclusion or reference.

This specification shall be published after qualification testing just prior to product delivery, and shall serve as the basic configuration control tool for the life cycle of the program.

## VERSION DESCRIPTION DOCUMENT

The Version Description Document (VDD) is the vehicle for informing its recipients of the complete configuration identification of a CPCl as it is constituted on a selected medium (tape disc, etc.) at the time of CPCl release to a user agency. The VDD provides explicit information regarding:

1. Identification of all elements (computer program components or equivalent) that collectively make up the CPCl.
2. Identification of all elements that were employed in the generation of the subject CPCl; this includes data that was used in the compilation of the object state, support software such as the compiler or assembler, and any other software elements that were used in the CPCl generation process but are not integral to the CPCl.
3. Description, in operational terms, of the changes that have been implemented in the release; these are changes from the previous release. For an initial release, this would be a summary of all ECPs that have been incorporated into the CPCl during its development leading up to its release.

## **DATA BASE SPECIFICATION**

The Data Base Specification describes and identifies all data in each data base. It specifies the system or systems using the data base, the storage requirements, and the physical description and organization of the data base. It references all support programs available for handling the data base. Labeling conventions are specified and the instructions for updating or modifying the data are provided. Each file, table, and item is described.

## **POSITIONAL/STATION/TACTICAL OPERATOR'S MANUAL**

The purpose of this manual is to provide operating procedures to an operator in relation to a particular position or station within, or as a part of, a larger computer-based tactical system. It will focus upon the specific equipment configuration and information processing functions carried out at that particular position/station and will concentrate on the following:

1. Data formats
2. Equipment functions
3. Procedures to be followed
4. Codes utilized
5. Electrical interfaces
6. Failure modes
7. Function/action controls
8. Display organization and presentation.

Individual positions/stations may include, as examples, the air defense system display console, tactical fire direction terminal, timesharing system terminal, message I/O device, etc. Key ingredients of the manual deal with equipment operation and procedures to initiate, receive, and process the flow of information for a particular station/position within the system.

## **COMPUTER OPERATOR'S MANUAL**

The operator's manual deals primarily with information required by the computer operator and supervisors engaged in system operation. The manual addresses system monitoring and actions necessary to commence and sustain system operation and reactions required to overcome failures encountered during processing.

The purpose of this manual is to provide the computer operator with instructions regarding the basic procedures for monitoring and operating the various equipments comprising the system, including the central processor, computer terminals, and other ancillary components. The manual will also provide procedural instructions for the following actions:

1. System start up/initiation
2. Testing
3. Restart after failure
4. Recovery from partial or component failure
5. Hardware reconfiguration
6. Maintenance and diagnosis of performance or error conditions
7. Logging operational events
8. Overall system management
9. Proper manipulation and interpretation of switches/displays/keys/controls.

## **SOFTWARE MAINTENANCE/PROGRAMMER'S PROCEDURES MANUAL**

The Software Maintenance/Programmer's Procedures Manual is used by a limited number of personnel assigned to the critical task of generating and maintaining the computer program library of the facility. The system generation/maintenance task is assigned to highly qualified systems/programmer personnel who can accept the responsibility for a closely monitored, carefully administered system regeneration/software maintenance cycle.

The Software Maintenance/Programmer's Procedures Manual describes procedures, input data structures, and control mechanisms used to generate or to modify computer software. It is intended to supplement the Programmer's Reference Manual, assumes the availability of compiled/assembled programs, and introduces the programmer to a system that describes the procedures for incorporating tested software into an operational library system or for introducing changes to data/instructions contained within existing library programs. Procedures for uniquely identifying programs and sequencing them properly into the library for future recall are also included in the document. Once this revised program has been incorporated into the library system, it can be loaded for testing, or additional modification, correction, or operational use.

## COMMAND/STAFF MANUAL

This manual is provided to interested military managers and staff officers who have a requirement to understand, in general terms, how the system functions, whether they are responsible for its operation or concerned with utilizing the products of its operation for mission accomplishment.

The purpose of the Command/Staff Manual is to provide an understanding, in general terms, of the purpose, scope, and functions performed by system hardware/software operating in concert. This manual will orient management and staff personnel with the overall system design capability and with the role it plays in the flow of information in a military environment. This manual must fulfill the critical objective of conveying system architecture to the generalist (laymen) who need not become familiar with highly detailed technical parameters, but with the overall contribution of a particular system to mission accomplishment.

This manual will generally be concerned with conveying the purpose of the system and its interaction with other existing systems; the performance of systems' functions with respect to hardware configuration, processes performed, and military utility; and lastly, the presentation in nontechnical terminology of the overall function of the system. Material presented must be sufficient to familiarize staff officers having a direct relationship with the system through billet assignment with the nature of the information or results produced by the system during normal operation, as well as familiarize them with the means by which this information is produced (i.e., sources and processes).

## **MAINTENANCE AND DIAGNOSTIC FAULT CATALOG**

The Maintenance and Diagnostic Fault Catalog contains the information necessary to isolate a hardware malfunction in a computer-based system to the lowest replaceable unit. It lists the stop numbers or error messages that are presented to the operator. These numbers (or messages) refer to one or more potentially faulty units which are identified by card number, corrector numbers, part number, or other nomenclature.

The Maintenance and Diagnostic Fault Catalog includes instructions for running the maintenance and diagnostic programs and for isolating the truly malfunctioning unit from a list of possibles.

## **SOFTWARE QUALITY ASSURANCE PLAN**

The Software Quality Assurance (SQA) Plan describes the organization and system of the contractor to assure that software and documentation delivered under the contract complies with the requirements of the contract. The SQA Plan shall be oriented toward the specification, design, and development of software and documentation that is planned and developed in consonance with the contractor's other administrative and technical programs.

The SQA Plan defines how the contractor will implement the quality assurance program as applied to a software project. It provides the Government with detailed knowledge of the contractor's SQA program and may be used to monitor the SQA program, as implemented.

## TEST PLAN

This document identifies and describes the tests that are proposed by the development contractor, the independent test contractor, or the Government test agency responsible for system testing. Among its purposes are to describe the tests in relationship to other related test programs, secure the necessary approval, control redundancy, and ensure that the necessary facilities, instrumentation, personnel, and test configurations will be available when needed. It also forms the basis for detailed test design, execution, and evaluation.

The format for a Test Plan will be consistent whether it was written for a system, subsystem CPCI, or even a module level test, but the scope and objectives of a particular plan will reflect the appropriate purpose of this test. The Test Plan shall identify all other tests that will be satisfactorily completed prior to the initiation of this test, and shall have sufficient background information, test philosophy, and concept of testing so that the detailed test requirements/specification document can unambiguously implement this plan.

The Test Plans will include at least the following general topics, and any other required to describe the specific system under test: scope, applicable documents, system identification, prerequisite testing, objectives, qualification requirements, criteria, testing methodologies, assumptions, limitations and constraints, external interfaces, schedule, location, responsibilities, and test controls.

## **TEST SPECIFICATION/REQUIREMENTS**

The Test Specification is a descriptive implementation of the appropriate Test Plan and is the document in which the basic test criteria are identified and the basic method to be used in the specific test is explained. The Test Specification will ensure that the necessary facilities, instrumentation, personnel, and test configuration are identified and will be available when needed and be the basis for drafting the detailed test procedures.

The formats of Test Specifications documents will be identical but the content will vary, depending upon the type of Test Plan that this Test Specification is implementing, i.e., system, subsystem, CPCI, or module.

The Test Specification in general is an event-oriented description of the test to be performed and is an expansion of the Government-approved Test Plan. The Test Specification should address the following general topics and any other required to describe the specific test being specified: scope, applicable documents, applicable prerequisite testing, personnel, facilities, equipment, support software, test case definitions, matrix of test cases to requirements, inputs, required accuracies, expected output, data collection method, error reporting, timing requirements, interface, degradation, and schedule.

## **TEST PROCEDURES**

The Test Procedures provide the detailed instructions for the execution of a test. The procedures are developed from the Test Specifications, Test Plans, and relevant design documents. The procedures will specify the test sequence of events and step-by-step operator actions. It shall describe the total equipment, manpower, and supporting documentation required to execute the tests. The Test Procedures should address the following general topics and others required to describe the specific procedures for this item: applicable documents, prerequisite testing, facilities, equipment, equipment initialization, support initialization, personnel stationing, data recording, safety precaution, error reporting, and test execution management.

## TEST REPORT

The purpose of the Test Report is to detail the results of the executed test. The Test Report will describe the purpose and nature of test and describe in detail any deviations from the Test Specification or procedures and will compare the test results with the expected output and requirement being tested. The test results should address the following general topics and other appropriate topics reflected in the Test Plan or Test Specification: introduction, applicable documents, summary, test objectives, test configuration, test deviations, matrix of test results vs. requirements, discrepancies of test item, evaluation criteria and analysis, and the plan for correction of deficiencies.

## TROUBLE REPORT

The Trouble Report is an information format that consists of identification data, problem descriptions, problem analysis, recommended actions, and corrective actions. Identification data identifies the field/test organization that discovered the problem, the system identification, the program identification, the date of report generation, the personnel who discovered or who can add information about the problem, and a contact point.

The problem description identifies the circumstances/environment when the problem was discovered, a description of the problem symptoms, the field operator's estimate of problem cause, any work-around problem definition actions taken and their results, and whatever data can be provided by the user to assist the maintenance/development activity in problem reproduction, analysis, and correction. Program analysis is a verification of the problems, a description of the cause, the impact of the problem on the software/systems, concluding impacts and identification, including contact point of the analyst. The recommended action describes the proposed corrective action, provides an estimate of the resources to implement the recommended action, and provides the person/authority granting the approval/disapproval with dates and signatures.

The action taken is a description of the actions taken to correct the problem, any pertinent information describing the implementation process, and a statement of/reference to the level of verification testing performed. The implementing activity/person is identified with dates and contact points as is the approving authority for the release of the correction.

The purpose of the Trouble Report is to provide a vehicle for field-user personnel and test personnel to record problem descriptions/symptoms and to provide the maintenance/development personnel with a means of describing analysis and corrective actions. The Trouble Report triggers a change process under configuration management control and is a component of a larger change process.

**PROCEEDINGS OF THE SOFTWARE WORKSHOP**

**JOINT LOGISTICS COMMANDERS**

**JOINT POLICY COORDINATING GROUP ON COMPUTER RESOURCE MANAGEMENT**

**MONTEREY, CA, 2-5 APRIL 1979**

**Report of the Panel on**

**STANDARDS FOR SOFTWARE QUALITY**

**Chairman: Robert Dunn  
ITT Avionics Division**

**Co-Chairman: Richard Maher  
TRW DSSG**

## OBJECTIVES

### 1. General

The panel's objectives were to define the content, to the level of detail practicable, of a set of documents that would establish expectations of software quality assurance programs useful to all three services. As will be seen in "Discussions," that set was defined to include:

1. Standard
2. DID for a software QA plan
3. Handbook
4. Guidebook.

In addition to these, other DIDs and an explanation of terms used in the field of software QA were also considered.

Each section of this report contains a paragraph on each of the members of the set.

### 2. Standard

Complete updating of existing draft revision of MIL-S-52779, dated 21 March 1979, with intent to secure triservices endorsement for consideration as a DOD-level standard.

### 3. DID for QA Plan

Define or develop a Data Item Description (DID) that would be in full compliance with the recommended Software Quality Assurance MIL-S-52779 as revised by the entire panel.

### 4. Handbook

Develop an approach, outline, and format for a Quality Assurance Handbook in support of the Standard.

### 5. Guidebook

Discuss the merits of developing a DOD Guidebook for Software Quality Assurance and the contents thereof.

## SCOPE

### 1. General

A single document, MIL-S-52779 (AD), enjoys considerable currency in all three services as a software QA standard. Nevertheless, it has not been considered entirely acceptable by all. It has eluded status as a triservice standard since its original 1974 publication, despite guidebooks published by both ASD and ESD, a pamphlet published by SAMSO, and a DID by NOSC, all referencing and supporting MIL-S-52779. In a sense competitive with, but not fundamentally inimical to, MIL-S-52779 is MIL-STD-1679 (NAVY) ("Weapon Systems Software Development"), which covers the elements of a QA program. Also, AF Regulation 800-14 ("Acquisition and Support Procedures for Computer Resources in Systems") may be modified to reflect the need for a QA program.

In brief, although the services nearly had a de facto triservice standard, there was none, with the consequence that there was no single way of fashioning a QA plan or explaining the implementation of one.

The panel restricted its interest to software embedded in weapons systems and excluded other software (e.g., EDP or computer-aided design). The panel did not, however, exclude software used directly in support of embedded software. Although, as noted by H. Tzudiker, chairman of QCIC-0001, MIL-S-52779 was engineered to assure its applicability to all kinds of software, the panel did not discuss it except within the scope of weapons systems.

In its discussions, the panel deliberately avoided any definitions of software QA activities tied to specific development and maintenance methodologies, to enable near-term triservice acceptance.

## 2. Standard

Areas considered included both assessment of other existing documentation pertinent to software quality assurance and the possible impacts to activities of other panels and subpanels. Consideration of firmware, while included in MIL-S-52779, was excluded from panel activities to the extent that no attempt was made to describe or define the term within the text of the document.

## 3. DID for QA Plan

An important part of MIL-S-52779 is the requirement that a Software Quality Assurance Plan (i.e., "the plan") be written and implemented by the software contractor. In order that such a Plan, if specified as a deliverable, be included in a Contract Data Deliverable List (CDDL) and to provide more specific information about the format and content of such a Plan, a DID compatible with MIL-S-52779 is required.

## 4. Handbook

DOD plant representatives and DCASR personnel have had a difficult time in evaluating contractor software QA plans, implementing QA programs, and monitoring results. This has been due in part to lack of well-defined and consistent requirements, differences in approaches of various DOD services programs, and availability of experienced DOD personnel. The situation has been further aggravated by software QA personnel guidance and training insufficiencies. The issuance of MIL-S-52779 will establish a solid, consistent software QA requirements baseline, but will require a handbook to assist government personnel in evaluating plans and establishing/monitoring the resultant in-plant QA programs. The task of the QA Handbook Subpanel was to review each requirement of MIL-S-52779 (as modified at this conference) and determine how the handbook could provide the necessary interpretation, application, and evaluation criteria assistance.

## 5. Guidebook

The move to develop a triservice standard accompanied by a DOD handbook to assist government agencies in evaluating quality assurance programs points out the need for DOD-level guidance and training to fully implement MIL-S-52779. This guidance and training were taken as the scope of the guidebook.

## APPROACH

### 1. General

Prior to the Workshop, each of the panel members was sent a package containing the following documents:

MIL-S-52779 (AD) (1974)  
MIL-S-52779 (AD) (November 1978 draft)  
MIL-STD-1679 (Navy) (pages applying to QA only)  
DI-R-2174, Navy, "Software QA Plan"  
TADSTAND 9, Navy, Software Quality Testing Criteria  
UDI-A-173, NOSC, Software QA Program Procedures  
UDI-A-171, NOSC, Software QA Policy  
UDI-A-170, NOSC, Software QA Program Organization  
UDI-A-172, NOSC, Software QA Program Practices  
ESD-TR-77-255, Acquisition Guidebook on Software QA  
ASD-TR-78-8, Software Acquisition Guidebook for QA

The panel members were requested to familiarize themselves with these. Other documents subsequently referenced were AF Reg. 800-14, SAMSO Pamphlet 74-2 ("Contractor Software QA Evaluation Guide"), and MIL-S-52779 (AD) March 1979 draft, which was distributed by H. Tzudiker at the meeting.

With these as background, the panel proceeded to identify the set of documents that would totally define software QA for weapons systems and the topics that would comprise the content of each. The top-level document, henceforth referred to as the "Standard," was discussed to the point where all substantive differences of philosophy were resolved. The panel then divided into four subpanels, each corresponding to one of the principal documents defined, to develop the document to the extent practicable. After a full day of subpanel deliberations, the full panel reconvened to hear the reports of each of the subpanels and to discuss each in an attempt to reach consensus on recommendations for each of the documents.

### 2. Standard

The activities were basically divided into four phases:

- a. Full panel review and critique of the existing 21 March 1979 draft revision provided by the QCIC-001 group
- b. Subpanel individual and group review and group resolution of proposed wording for each paragraph contained in the resultant final draft
- c. Coordination of proposed changes with panels or subpanels which might be affected. This activity also included the solicitation and consideration of comments and change proposals from other panels and individuals
- d. Final full panel review and critique of the subpanel final draft.

In the course of subpanel activity, the pertinent sections (5.9, 5.10, 5.11) of MIL-STD-1679 (Navy), dated 1 December 1978, were received for consistency or conflict with the MIL-S-52779 draft. No inconsistencies or conflict were identified.

### 3. DID for QA Plan

The subpanel listed and discussed existing DIDs applicable to software quality assurance efforts. Included in this list are:

- a. DI-A-XXXX Computer Software Quality Assurance Plan
- b. DI-R-XXXX (Navy) Computer Software Quality Assurance Program Plan
- c. DI-R-30510 (USAF) Quality Program Plan
- d. UDI-A-170 (NOSC) Computer Software Quality Assurance Program Organization
- e. UDI-A-171 (NOSC) Computer Software Quality Assurance Policy
- f. UDI-A-172 (NOSC) Computer Software Quality Assurance Practices
- g. UDI-A-173 (NOSC) Computer Software Quality Assurance Procedures
- h. UDI-R-111A Software Quality Control Plan
- i. UDI-R-21374A (Navy-AS) Plan, Quality Assurance Program
- j. DI-R-2174 Software Quality Assurance Plan

The subpanel was unanimous that DI-R-2174, written to be compliant with MIL-STD-1679 (Navy), was the most appropriate basis for a MIL-S-52779 compliant DID. An outline was made of each point in MIL-S-52779 that must be covered in the DID. These outlines were compared and the necessary additions and deletions to achieve a MIL-S-52779 DID were identified. With this information, the subpanel wrote the new required section. Each section of DI-R-2174 that was to be retained was reviewed and rewritten to make it compatible with MIL-S-52779. A full draft (Appendix 4) of the proposed new DID was then assembled. This draft was reviewed by the entire quality assurance panel after the final review of the ultimate MIL-S-52779 revisions and refinements. Further changes to the proposed DID were made during this review. (Time limitations permitted consideration only of issues of substance. From an editorial point of view, the DID must be regarded as a first draft).

### 4. Handbook

The subpanel agreed with the previous OSD direction to revise and expand SAMSOP PAMPHLET 74-2 as a DOD QA Handbook. The style and format of SAMSOP 74-2 was adopted for the QA Handbook. Indeed, the subpanel's approach was to provide suggestions to A. Pond of SAMSOP/PMGQ, who is performing the revision of 74-2 under an OSD task, and was also a subpanel member.

It was agreed that QA Guidebooks containing general guidance/tutorial information are required to assist in the use of the Handbook. By the same token, it was agreed that much of the information in the current Air Force ASD and ESD Software Quality Assurance Guidebooks is directly transferable to the QA Handbook. During the subpanel working sessions, requirements clarification questions and suggested changes were jointly reviewed and coordinated with the Standard, QA Data Item Description, and QA Software Guidebooks subpanels. The Handbook will be written to apply to all phases of life-cycle quality assurance where MIL-S-52779 is invoked. However, it was not clear how government organic software support agencies would apply the Handbook for Post Deployment Support Systems (PDSS). This problem was not specifically addressed by the subpanel, but it was felt that PDSS should invoke a direct transfer of MIL-S-52779 requirements and disciplines. In this case, the QA Handbook has direct applicability for PDSS.

The approach adopted by the subpanel was to follow MIL-S-52779 exactly to provide a corresponding Handbook to Paragraph A, B, and C for each stated requirement. The Handbook paragraphs will define:

Paragraph A - Review of Requirements - A review of the stated MIL-S-52779 requirement including any clarification and/or interpretation

Paragraph B - Application - A discussion of application, scope, and practices, including examples of how the requirements applied to Government programs

Paragraph C - Criteria for Evaluation - Specific checklists, questions, and other information to aid in analysis/evaluation of QA plans, programs, and QA program results.

## 5. Guidebook

The subpanel reviewed existing Air Force Systems Command Software Quality Assurance Guidebooks for applicability to a guidebook reflecting the Standard.

## DISCUSSION

### 1. General

The minimum set of documents considered necessary to govern the QA aspects of weapons systems software were defined as:

- a. Standard
- b. DID for software QA plan
- c. Handbook to help interpret Standard in evaluating QA programs
- d. Guidebook to provide further background in software QA.

In short order, MIL-S-52779 (AD) March 1979 draft was accepted as the basis for the Standard, since this version had been drafted by Working Group QCIC 001 under the DOD Standardization Program as chartered by DARCOM, and since MIL-S-52779 (AD) met the criterion of independence from any specific development methodology. There were, however, significant reservations concerning the lack of reference to the monitoring of work certification procedures, the distinctions between program library control and the control of documentation, the inclusion of references to contractors' programming conventions and practices and compliance with them under the documentation requirements, and a requirement to collect error data and submit such data to a common data base.

These and several lesser matters were discussed partly by the Standards subpanel and partly by the panel as a whole, resulting in the draft of Appendix 3.

In the matter of collecting error data, the panel felt that, as commendable and useful as a common error experience data bank is (such as the one developed by RADC), it was not practical to impose the cost penalty for this on each program through the use of a QA Standard.

The panel identified several aspects of software QA that should be amplified beyond the extent attainable in the Standard. These items were:

- a. Post Delivery QA
- b. Records
- c. Cost
- d. Independence of the quality responsibility
- e. Error data analysis notwithstanding Paragraph 3.2.9 ("Corrective Action") of the draft standard of Appendix C, Dr. Schneidewind would like it noted that he feels software error data collection and analysis is an integral part of software QA and should be mandated on embedded computer projects by being incorporated in a QA Standard.
- f. Transition from development to deployment
- g. Documentation substantiation (traceability)
- h. QA tools
- i. Definition of terms used in software QA.

With the exception of the last of these, for which a specific recommendation was made, it was the panel's opinion that amplification of these matters be handled in the Handbook and the Guidebook, as appropriate.

Regarding the first item on the list, it was also noted that the Guidebook should contain a tutorial on software QA throughout the life cycle, since MIL-S-52779 is also intended to apply to any contracts for post-delivery program modifications as well as for initial acquisition.

## 2. Standard

The issues, contributing factors/constraints, etc., which were of concern to the subpanel, and the decisions made, ultimately fell into two categories: level of detail and scope of applicability, detailed below.

Based on these listed items, the following conclusions were reached in subpanel discussions and are reflected in the draft document (Appendix 3):

- a. The proposed standard should use generic terminology where practical, be held to the minimum level of detail required to adequately state the requirements while minimizing the capability for misinterpretation, and include all categories of software which may affect the quality of end item products.

- b. The proposed standard should be applicable, when imposed by contract, to all phases of the life cycle, be mandatory for all development/acquisition activity and contracted operational maintenance activity, and serve as a basis for quality assurance of organically conducted operational maintenance activity.
- c. The proposed standard should specifically address the subject of tailoring the program to suit the needs of the procurement.

Panel concerns relating primarily to level of detail were:

- a. Infringement on contractor organizational prerogatives
- b. Intrusion into areas of requirements covered by other DOD documentation, specifications, and standards
- c. Imposition of unrealistic implementation detail in terms of methodologies and tools, and the levels of control and documentation
- d. Availability of other documentation to provide guidance and direction for specific procurement activities (i.e., data item description, guidebooks, handbook, specifications and standards, and individual procurement documents)
- e. Identification of any differences in requirements associated with end item vs. nondeliverable software or with use categories (e.g., test, support, operational)
- f. Terminology and nomenclature acceptable for triservices endorsement, broad enough for general application, and specific enough to minimize misinterpretation.

Panel concerns relating to scope and applicability were:

- a. Life-cycle usage and alternatives for the post-delivery operational maintenance phase
- b. Need for assurance of tailoring to meet the differing requirements/needs of individual procurement effort.

### 3. DID for QA Plan

It was agreed early that the software quality assurance function should apply not only to the final, delivered computer program but also to the documentation that is produced during the development of that program and which supports it.

Much of the discussion during the drafting of the MIL-S-52779 compliant DID centered around the instructions that could be given to the contractor through the DID, either directly or implied, about the software assurance actions, organization, and scope. The approach taken was not to give specific direction, which it was felt would be inappropriate in the DID, but rather to require the contractor to describe what he will do with regard to these actions, organization, and scope. Thus, a Software Quality Assurance Plan could state "none" in some sections if the contractor did not plan to perform certain actions; the contracting agency could evaluate this response to see if such an answer were sufficient or appropriate for an acceptable plan in the particular circumstances.

A disagreement was based on whether the Plan should describe only the procedures done by the Software Quality Assurance "function," or if it should also include the procedures done by the software developers that support or accomplish the quality assurance objectives. This disagreement was resolved by asking in the DID for the contractor to state in the Plan both what the quality assurance function does and what the developers do to support or accomplish the quality assurance objectives.

It was felt that the Plan should address the role of the quality assurance function in the final certification of the software; such a section has been added to the draft DID.

There was some objection to specific words in the DID, particularly the word "testability," because there is no agreement on or capability for their precise definition or measurement. This discussion emphasized the need for a glossary or definition of terms as might result from an update and expansion of MIL-STD-109B, Quality Assurance Terms Definitions, to include software definitions and terms.

It was stated that the Air Force could not require the delivery of a Software Quality Assurance Plan because of existing Air Force regulations. However, waivers have been obtained. Discouraging the delivery of Software Quality Assurance Plans was not considered appropriate and consideration should be given to reversing any such Air Force regulations.

The subpanel also discussed the requirement for a DID for error or problem reporting. Such a DID might enable the more effective collection, analysis, and application of software error data across projects. Several existing DIDs were identified and reviewed that appear to fulfill this need. Questions concerning the appropriateness and cost of requiring such error reporting from the software development contractor were raised.

#### 4. Handbook

##### Handbook Philosophy

The first activity was a philosophical discussion of purpose, intent, and format of the QA Handbook. A brief summary follows:

- a. A set of Quality Assurance Requirements (MIL-S-52779) and a Data Item Description lead to an approved Quality Assurance Plan and implementation of a Quality Assurance Program. Software quality assurance "freedom of action" requirements led to an organizational structure including quality assurance and configuration management functions. These functions involve various processes, test programs, analyses, and a series of overlaying monitoring programs. The Handbook provides specific information, guidance, and checklists to assist the Government and/or contractor in planning, implementing, or monitoring Quality Assurance Programs. The Handbook tracks each requirement of MIL-S-52779 and must address their implementation across all of the interfaces in a software development activity. The Handbook is organized to assist in interpretation/translation of requirements, application in planning and conducting QA programs, and monitoring/evaluation results. This broad application for Handbook utilization was adopted in establishing the QA Handbook subpanel's approach to Handbook preparation.

- b. The prime purpose is to assist government and prime contractor personnel in (1) evaluation of QA proposals, programs, and plans, (2) implementation of QA programs, and (3) monitoring, analysis, and evaluation of QA program activities and results.
- c. The scope is to include all phases of DOD weapons system software development where MIL-S-52779 is invoked. The interface and application of the Handbook in PDSS is only applicable where MIL-S-52779 is the controlling requirement.
- d. The Handbook style and format will follow MIL-S-52779 exactly as well as DOD standardization requirements for preparation of handbooks. Each MIL-S-52779 paragraph will have a corresponding number paragraph in the Handbook and will contain in a section or detailed subsections a paragraph:
  - A. Review of Requirements
  - B. Application
  - C. Criteria for Evaluation
- e. The Handbook will interface with and rely on a set of Guidebooks to provide the necessary background information, tutorial instruction, and guidance for use of the Handbook in specific areas of application and interpretation.

#### Organizational "Independence"

It was agreed that the "Freedom of Action" requirements in MIL-S-52779 provide for adequate organization independence. The handbook and particularly the QA Guidebooks should elaborate, state examples, and discuss interfaces for both software development contractor and subcontractor organizations.

#### SAMSO Pamphlet 74-2 Review

SAMSOP 74-2 was reviewed for content, style, and applicability as a starting point:

- a. SAMSO will revise the pamphlet to become a DOD Software Quality Assurance Handbook.
- b. The first release will be 180 days after MIL-S-52779. This is a current SAMSO/PMGQ commitment.
- c. The subpanel will participate in the generation and/or revision process.
- d. The current use of SAMSOP 74-2 is primarily for evaluation of software quality assurance in conjunction with AF/SAMSO RFP review.
- e. A significant expansion of Section C, "Criteria for Evaluation," will be required for QA Handbook use.
- f. Subsections will be established in the Handbook to elaborate and detail complex paragraphs of MIL-S-52779.
- g. QA Guidebooks and training courses must be established to augment and guide the Handbook user.

### MIL-STD-1679 (N) Review

The subpanel reviewed MIL-STD-1679 (N) for comparison with MIL-S-52779 as a source for Handbook/Guidebook material. It was agreed the MIL-S-52779 covers all the requirements stated in MIL-STD-1679 (N). MIL-STD-1679 (N) does contain a great deal of well written information suitable for QA Handbooks and/or Guidebooks and will be included, where appropriate.

### MIL-S-52779 Review of Handbook Requirements

The following comments/actions and results are listed by specific MIL-S-52779 paragraph references:

- a. Paragraph 3.1, Software QA Program - This requirement is very complex as written and will be covered in the Handbook in three separate subsections:
  - (1) Program Definition
  - (2) QA Plans, Content, Scope, and Style
  - (3) Authority and Organizational Freedom.

Each subsection will contain detailed user information and evaluation criteria.

- b. Paragraph 3.2.1., Tools, Techniques, Methodologies, and Records - The Handbook will include each topic as a separate subsection and include error analysis information under Tools or Methodologies.
- c. Paragraph 3.2.3, Work Certification - The Handbook will place emphasis on "resource allocation" in this requirements section. The goal is to assure a major concentration of contractor QA resources on high-leverage, "up-front" software development activities. Focus on resource allocation will also provide a significant overall level of effort during the entire software acquisition process. It is intended that "work certification" be more than just a formalized, witness function at the end of a series of activities which may or may not be supported by adequate QA resources. The main payoff is early involvement with sufficient resources for QA.
- d. Paragraph 3.2.4, Software Documentation and Programming Conventions - The Handbook will treat the general subject of Documentation and Programming Standards and/or Conventions as separate issues. Emphasis will be placed on:
  - (1) Methods of showing correctness and completeness of documentation
  - (2) Auditing function for showing compliance to programming/coding conventions and standards.

The QA importance of adhering to standards and conventions is worthy of special, detailed treatment in the Handbook and will be included in Criteria for Evaluation.

- e. Paragraph 3.2.6, Reviews and Audits - The Section C, Criteria for Evaluation, will contain detailed information on each major milestone such as SDR, PDR, CDR, etc. Much of the existing information in the ASD and ESD Quality Assurance Guidebook (TRW and SDC) will be used to develop this section.
- f. Paragraph 3.2.7, Configuration Management - An in-depth discussion of the relationship between quality assurance and configuration management will be included in this section of the Handbook. These areas must complement each other in an appropriate management structure to execute an effective QA program.
- g. Paragraph 3.2.9 (c), Corrective Action - A lengthy discussion was held concerning requirements for "corrective action" and how determination could be made by analytic means. Error analysis should be a primary means of evaluating QA programs and directing attention to specific needs for correction action or recovery programs. It was agreed that use of "error analysis" should be described in the Handbook and treated in-depth in the QA Guidebooks. It was also agreed that much theoretical and practical effort is required before software error definition, reporting categories, and/or analysis is mature enough to assist in conducting and monitoring QA programs.

## 5. Guidebook

Initially, the subpanel members were sharply divided over the need for any guidebooks, much less a DOD-level guidebook on software quality assurance. The ensuing discussion revealed that the principal objection centered more on the difficulty of producing and coordinating a triservice document than on the need for the document. Additional objections centered on the perceived uniqueness of the various services requirements and the possible constraints inherent in the development and imposed use of a triservice document.

Examination of the content of current quality assurance Guidebooks and restatement of the intended treatment of quality assurance topics in the proposed DOD Guidebook seemed to satisfy even the most skeptical.

Concurrent with and interspersed throughout the discussion of the need and content of a DOD-level guidebook was the need for a DOD-level training course for quality assurance personnel. The objection to such a course followed the same theme of service uniqueness but in this case also centered on which agency could best conduct such a course (one subpanel member was already conducting quality assurance training but on a much smaller scale than would be required for a triservice course).

In summary, the subpanel deliberations resulted in identification of a need for a DOD-level multiservice Software Quality Assurance Guidebook. The purpose of the Guidebook is to encourage standardization by providing textual material of direct interest to management, contracting, and engineering personnel. The Guidebook would be based upon existing QA and related guidebooks. It would discuss and elaborate on the topics contained in MIL-S-52779, the implementation of the QA program throughout the life cycle and, along with MIL-S-52779, would become the basic text for a standard DOD-level software quality assurance training course. The training course would be tailored to the specific requirements of acquisition managers, design agents (software developers), quality assurance personnel, and contract administration services (CAS). An outline of the major topics in the proposed DOD Guidebook is presented here.

**TITLE: SOFTWARE QUALITY ASSURANCE PROGRAM GUIDEBOOK**

**I INTRODUCTION**

**APPLICABILITY**

**II APPLICABLE DOCUMENTS**

**DEFINITION OF TERMS**

**III SQA PROGRAM REQUIREMENTS**

**DISCUSSION OF TOPICS IN SECTION 3 MIL-S-52779**

**IV SQA PROGRAM IMPLEMENTATION**

**DISCUSSION OF SQA PROGRAM ACTIVITIES THROUGHOUT THE LIFE CYCLE**

**Note:** Guidebook to be based upon current guidebooks developed by Air Force Systems Command.

**RECOMMENDATIONS**

**1. General**

The Panel recommended that a glossary of software QA terminology, applicable to all software QA documents, be developed and appended to MIL-STD-109.

**2. Standard**

Based on the conclusions of Paragraph 4.2 and the assumption of triservice endorsement of the MIL-S-52779 draft update, dated 5 April 1979 (Appendix 3), the panel recommendations for JLC action are:

- a. Complete necessary steps for issuance of the draft update as a DOD Standard
- b. Although there was no time to discuss this with the panel at large, the subpanel also recommended:
  - (1) Establish a mechanism (standing committee, ad hoc groups, etc.) for review of overall progress of software quality standards efforts, pursuing resolution of unresolved quality concerns and providing additional recommendations for changes necessary to develop and maintain a quality assurance activity compatible with state-of-the-art advancements in software technology.
  - (2) To evaluate the necessity or desirability of a course of action aimed at consolidating comparable activities of other agencies, associations, and professional groups for development of a commercial/DOD compatible set of standards, specifications, and guidance documentation.

3. DID for QA Plan

The panel recommends that the attached draft DID (Appendix 4) be reviewed, approved, and issued to support the implementation of MIL-S-52779.

4. Handbook

The panel recommends that the Handbook being prepared by A. Pond under OSD Task to update SAMSO Pamphlet 74-2, with the modifications suggested by this panel, be approved and issued as a triservice document. (Mr. Pond intends to send a draft of the Handbook, when available, to the subpanel members for their review.)

The panel also recommends that the JLC support contributing activities to define, refine, and generally develop error analysis as a prime means of flagging corrective action and evaluating QA programs.

5. Guidebook

The panel recommends that the JLC request OSD QA Council to issue Task for Development of DOD-Level Software QA Guidebook.

The panel recommends that the JLC establish a DOD Software QA Course within the Defense Management Education Training Concept.

APPENDIX 1 - PARTICIPANTS

## APPENDIX 1 - PARTICIPANTS

Chairman - R. Dunn, ITT Avionics Division  
100 Kingsland Road, Clifton, NJ 07014

### **Subpanel on Standards:**

Chairman - Mr. W. J. (Jack) Troube  
Space Systems Division Product Assurance  
Lockheed Missiles & Space Co., Inc.  
1111 Lockheed Way  
Sunnyvale, CA 94086

Members - Major Roy F. Miller  
Box 504  
LMSC/AFFPRO Det. 13/AFCMD  
Sunnyvale, CA 94086

- Mr. Earl B. Stewart  
U. S. Army Missile Research and Development Command  
Attn: DRDMI-QEA  
Redstone Arsenal, AL 35089

- Mr. Harvey Tzudiker  
U. S. Army Computer Systems Command  
Attn: Stop C100  
Fort Belvoir, VA 22060

### **Subpanel on DID for QA Plan:**

Chairman - Mr. Raymond J. Rubey  
Softech  
Claypool Building  
Suite 355  
4130 Linden Avenue  
Dayton, OH 45432

Members - Mr. James Cellini  
RADC/ISIS  
Griffiss AFB, NY 13441

- Mr. Clell Gladson - QT  
Defense Contract Administration Service  
Los Angeles Region  
11099 South La Cienega Boulevard  
Los Angeles, CA 90045

**Subpanel on DID for QA Plan: (cont)**

- Members**
- Mr. Calvin Showalter  
Naval Air Systems Command  
ATTN: AIR 53311  
Washington, DC 20361
  - Dr. Norman F. Schneidewind  
Professor of Information Science  
Department of Computer Science  
Code 52 SS  
U.S. Naval Postgraduate School  
Monterey, CA 93940
  - Mr. Melvin Mitchell  
SM-ALC/MMECM  
McClellan AFB, CA 95652

**Subpanel on Handbook:**

- Chairman**
- Mr. Richard A. Maher, 55/2856  
TRW Defense and Space Systems Group  
One Space Park  
Redondo Beach, CA 90278
- Members**
- Mr. Andrew Pond  
HQ, SAMSO/PMGQ  
P.O. Box 92960 - Worldway Postal Center  
Los Angeles, CA 90009
  - Mr. Michael Kirchner  
Defense Logistics Agency  
Attn: DLA-QES  
Cameron Station  
Alexandria, VA 22314
  - Mr. Tye Gibson  
HQ. Naval Material Command  
Attn: 08E3  
Washington, DC

**Subpanel on Guidebook:**

- Chairman**
- Mr. L.D. Parriott  
TRW, DSSG  
Mail Stop 55/3569  
One Space Park  
Redondo Beach, CA 90278

**Subpanel on Guidebook: (cont)**

- Members**
- Mr. Robert Dubois  
U.S. Army Materiel Development  
and Readiness Command  
DRCM-QA-E  
5001 Eisenhower Avenue  
Alexandria, VA 22333
  - Mr. P.A. Rhodes  
Code 9133  
Naval Ocean Systems Center  
San Diego, CA 92152
  - Mr. Clare E. Skrukrud  
SAI Comsystems Corporation  
P.O. Box A-81126  
2801 Camino del Rio South  
San Diego, CA 92138
  - Major David Austin  
HQ AFSC/PMN  
Andrews AFB  
Washington, DC 20334

**Panel Member At Large:**

F. Barricelli, COR ADCOM  
Attn: DRDCO-PT-J  
Fort Monmouth, NJ 07703

APPENDIX 2 - BIBLIOGRAPHY

## APPENDIX 2 - BIBLIOGRAPHY

1. MIL-S-52779 (AD) Software Quality Assurance Requirements  
April '74  
November '78 draft update  
March '79 draft update
2. MIL-STD-1679 (Navy) Weapons Systems Software Development
3. DI-R-2174, Navy, "Software QA Plan"
4. TADSTAND 9, Navy, Software Quality Testing Criteria
5. UDI-A-173, NOSC, Software QA Program Procedures
6. UDI-A-171, NOSC, Software QA Policy
7. UDI-A-170, NOSC, Software QA Program Organization
8. UDI-A-172, NOSC, Software QA Program Practices
9. ESD-TR-77-255, Acquisition Guidebook on Software QA
10. ASD-TR-78-8, Software Acquisition Guidebook for QA
11. DI-A-XXXX Computer Software Quality Assurance Plan
12. DI-R-XXXX (Navy) Computer Software Quality Assurance Program Plan
13. DI-R-30510 (USAF) Quality Program Plan
14. UDI-R-111A Software Quality Control Plan
15. UDI-R-21374A (Navy-AS) Plan, Quality Assurance Program
16. AF Reg. 800-14, Management of Computer Resources in Systems
17. SAMSO Pamphlet 74-2, Contractor Software QA Evaluation Guide

APPENDIX 3 - MILITARY SPECIFICATION  
SOFTWARE QUALITY ASSURANCE  
PROGRAM REQUIREMENTS

**APPENDIX 3 - MILITARY SPECIFICATION SOFTWARE QUALITY  
ASSURANCE PROGRAM REQUIREMENTS**

DRAFT  
MIL-S-52779  
5 April 1979

This specification is approved for use by all departments and agencies of the Department of Defense.

**1.0 SCOPE**

**1.1 APPLICABILITY**

When referenced in the item specification, contract, or order, this specification shall apply to the acquisition of software (computer programs, related data, and documentation) where the acquisition involves either software alone or software as a portion of a system or subsystem. This specification shall also apply to nondeliverable design, test, support, and operational software developed under the contract, unless specifically exempted. For purposes of this specification, the term software includes firmware.

**1.2 CONTRACTUAL INTENT**

This specification requires the establishment and implementation of a Software Quality Assurance (SQA) Program (hereafter referred to as the Program) by the contractor. The purpose of the Program is to assure that software developed, acquired, or otherwise provided under the contract complies with the requirements of the contract. It is intended that the program be effectively tailored and economically planned and developed in consonance with, or as an extension of, the contractor's other quality assurance, administrative, and technical programs. The term Program, as used herein, identifies the collective requirements of the specification. The Program shall require periodic assessment and, where necessary, realignment of the Program to conform to changes in the acquisition program. The Program is subject to disapproval by the Government whenever it does not accomplish the requirements of this specification.

**1.3 RELATION TO OTHER CONTRACT REQUIREMENTS**

The contractor is responsible for compliance with all provisions of the contract and for furnishing specified software which complies with all the requirements of the contract. The SQA Program Plan shall reference other plans; e.g., configuration management, test, development, etc., specified under the contract and shall be compatible and consistent with them and not unnecessarily duplicate their provisions. If any inconsistency exists between the terms of the contract and this specification, the Order of Precedence clause of the contract shall govern.

## **2.0 APPLICABLE DOCUMENTS**

### **2.1 AMENDMENTS AND REVISIONS**

Whenever this specification is amended or revised subsequent to its contractually effective date, the contractor may follow, or authorize his subcontractors to follow, the amended or revised document, provided no impact on schedule or increase in cost, price, or fee is required. The contractor shall not be required to follow the amended or revised document except as a formally authorized modification to the contract. If the contractor elects to follow the amended or revised document, he shall notify the contracting officer in writing of this election. When the contractor elects to follow the provisions of an amendment or revision, he must follow them in full.

### **2.2 ORDERING GOVERNMENT DOCUMENTS**

Copies of specifications, standards, and documentation required by contractors in connection with specific procurements may be obtained from the procuring agency, or as otherwise directed by the contracting officer.

## **3.0 REQUIREMENTS**

### **3.1 SOFTWARE QA PROGRAM**

Upon contract award, the contractor shall plan, develop, and implement an SQA Program which includes practices and procedures to assure compliance with all software requirements of the contract. The Program activities shall be a part of the management reporting system throughout the life of the contract. The contractor shall document the Program in the form of an SQA Plan (hereafter referred to as the Plan) which meets the requirements of this specification. The Plan shall identify organizational responsibilities and authorities for its execution and the events critical to its implementation. The Plan also shall identify and make timely provisions for special needs (controls, tools, facilities, skills, etc.) required for the Program and shall provide for detection, reporting, analysis, and correction of software problems and deficiencies. Contractor personnel performing quality functions shall have the responsibility, authority, and organizational freedom to evaluate software activities, identify problems, and initiate or recommend corrective action.

### **3.2 SOFTWARE QA PROGRAM REQUIREMENTS**

The Plan shall address the following requirements:

#### **3.2.1 Tools, Techniques, and Methodologies**

The Plan shall identify the tools, techniques, methodologies, and records to be employed in the performance of the work which will support QA objectives and describe how their use will augment or satisfy QA Program requirements. Examples include: operation research - systems analysis techniques, functional and performance requirements analysis, error analysis, software optimization tools, specification tracing, and coding conventions.

### 3.2.2 Computer Program Design

The Plan shall reference or document the procedures by which design documentation is reviewed to evaluate design logic, fulfillment of requirements, completeness, and compliance with specified standards. Design documentation shall be subjected to independent review prior to its release for coding.

### 3.2.3 Work Certification

The Plan shall reference or document the contractor's procedures for formally approving and certifying the description, authorization, and completion of work performed under the contract. The Program shall require monitoring to assure compliance with these procedures.

### 3.2.4 Documentation

Documentation standards and programming conventions and practices to be used for all software shall be referenced or documented in the Plan. The Plan shall reference or document the procedures to be applied to assure compliance with standards, practices, and conventions and delivery of correct documentation and change information to the Government. In addition, the Plan shall provide for the independent review of documentation and designation of contractor approval authority.

### 3.2.5 Computer Program Library Controls

The Plan shall reference or document the contractor's procedures and controls for the handling of source and object codes and related data in their various forms and versions, from the time of their initial approval or acceptance until they have been incorporated into the final media. The objective of these controls is to ensure that different computer program versions are accurately identified and documented, that no unauthorized modifications are made, that all approved modifications are properly incorporated, and that software submitted for testing is the correct version.

### 3.2.6 Reviews and Audits

The Plan shall reference or document the contractor's procedures for preparation and execution of reviews and audits, for establishing the traceability of initial contract requirements through the successive baselines, and for ensuring that the reviews and audits are conducted in accordance with the prescribed procedures. The schedule for review and audits shall be referenced or stated in the Plan.

### 3.2.7 Configuration Management (CM)

The Plan shall specify the relationships between the SQA and CM programs and shall reference or document the procedures for assuring that the objectives of the CM program are being attained.

### **3.2.8 Testing**

The Plan shall reference or document procedures for assuring the accomplishment of the following:

1. Analysis of software requirements to determine testability
2. Review of test requirements and criteria for adequacy, feasibility, and traceability and satisfaction of requirements
3. Review of test plans, procedures, and specifications for compliance with contractual requirements and to ensure that all authorized and only authorized changes are implemented
4. Verification that tests are conducted in accordance with approved test plans and procedures
5. Certification that test results are the actual findings
6. Review and certification of test reports
7. Ensuring that test-related media and documentation are maintained to allow repeatability of tests
8. The contractor shall ensure that support software and computer hardware used to develop and test software and hardware under the contract are acceptable to the Government.

### **3.2.9 Corrective Action**

The Plan shall reference or document procedures which assure the prompt detection, documentation, and correction of software problems and deficiencies. Procedures shall include:

1. Documenting and reporting problems and deficiencies to appropriate management levels
2. Analysis of data and examination of problem and deficiency reports to determine their extent and causes
3. Analysis of trends in performance of work to prevent the development of noncompliant products
4. Review of corrective measures to ensure that problems and deficiencies have been resolved and correctly reflected in the appropriate documents
5. Analysis or review as otherwise provided for in the contract.

### **3.3 SUBCONTRACTOR CONTROL**

The Plan shall reference or document the procedures to assure that all software acquired from subcontractors conforms to applicable requirements of the contract and this specification. When the Government elects to perform reviews at the subcontractor's facilities, such reviews shall not be used by contractors as evidence of effective control of quality of subcontractors by the contractor. It does not relieve the contractor of his responsibility for furnishing software that meets all contract requirements.

## **4.0 RESPONSIBILITIES**

### **4.1 CONTRACTOR**

Nothing specified herein relieves the contractor from the obligation to submit to the Government for acceptance end products that conform to all contract requirements.

### **4.2 GOVERNMENT REVIEW AT CONTRACTOR, SUBCONTRACTOR, OR VENDER FACILITIES**

The Government reserves the right to review, at their sources, all products or services, including those not developed or performed at the contractor's facility, to determine the conformance of products or services with contract requirements.

## **5.0 PREPARATION FOR DELIVERY**

The Plan shall reference or document procedures for assuring integrity of software products during handling, storage, preservation, packaging, and shipping.

## **6.0 NOTES**

(The following information is provided solely for guidance in using this specification. It has no contractual significance).

### **6.1 INTENDED USE**

This document will apply specifically to the acquisition of computer software where the acquisition involves either software alone, or software as a portion of a system or subsystem.

### **6.2 ORDERING DATA**

The procuring activity shall consider specifying the following:

#### **6.2.1 Procurement Requirements**

1. Title, number, and date of this specification
2. Software QA Program Plan. Consideration should be given to requiring the contractor to deliver a Software QA Program Plan in response to the invitation for bid, or request for proposal, or request for quotation and as a Contract Data Requirements List item (see Paragraph 6.2.2). The Plan should define the methods and procedures which the contractor proposes to use in fulfilling the requirements of this specification. Note: The Software QA Program Plan may be included as part of other plans; see Paragraph 1.3.

3. The application of this specification should be carefully tailored to meet the minimal essential needs of the acquisition.
4. Consideration should be given to citing current standards and specifications for configuration management, documentation, review, audit, development practices, work breakdown structures, etc.
5. Application of this specification to software maintenance contracts is encouraged.
6. Rapidly changing technology may require the acquiring activity to clarify use or application of the term "firmware."

#### 6.2.2 Contract Data Requirements

All plans, documentation, and reports which are required to be delivered to the Government will be specified on a DD Form 1423, Contract Data Requirements List (CDRL) or authorized equivalent. The format, type of copy, number of copies, degree of detail required, delivery schedules, and purpose of submission should be specified on the CDRL.

APPENDIX 4 - SOFTWARE QUALITY ASSURANCE PLAN

## APPENDIX 4 - SOFTWARE QUALITY ASSURANCE PLAN

DOD DI  
MIL-S-52779

- 3.1 The Quality Assurance Plan describes the organization and system of the contractor to assure that software and documentation delivered under the contract complies with the requirements of the contract. The Quality Assurance Plan shall be oriented toward the specification, design, and development of software and documentation that is planned and developed in consonance with the contractor's other administrative and technical programs.
- 7.1 The Software Quality Assurance Plan defines how the contractor will implement the quality assurance program as applied to a software project. It provides the Government with detailed knowledge of the contractor's quality assurance program and may be used to monitor the QA program, as implemented.
- 10.1 Unless otherwise stated in the solicitation, the effective date of the document(s) cited in this block shall be that listed in the issue of the DOD Index of Specifications and Standards (DODISS) and the supplements thereto specified in the solicitation and will form a part of this Data Item Description to the extent defined within.
- 10.2 The Software Quality Assurance Plan shall include all the subjects and items included below, if applicable.
  - 10.2.1 Introduction. Includes scope of QA Plan and identification of all development procedures and products subject to the provisions of the plan. The contractor policies, the objectives of the QA Plan, and the goals of the QA effort shall be stated. A list of applicable documents, their sources, and dates of issue shall be provided. The list shall include, but not be restricted to, Government standards and specifications applicable to the effort; the contractors' own standards, practices, and procedures; configuration management plans; software development plans; statements of work; and other governing specifications.
  - 10.2.2 Organization. Describe the organization of the QA function designed to comply with the requirements for quality assurance in MIL-S-52779. Include a chart showing the relationship of the QA function to management and other organizational elements. Describe the general authority and responsibility of the QA function and the means to be provided to exercise this authority, including signature approval/disapproval authority and procedures. The organizational dependence or independence of the QA function relative to other organizational elements shall be clearly described.

- 10.2.3 **Quality Assurance Procedures.** Describe QA procedures in the areas listed below which will support quality assurance objectives. In each area, identify all QA practices, techniques, and methodologies and describe how their use will augment or satisfy the QA requirements of MIL-S-52779.
- 10.2.3.1 Software Development. The Software QA Plan shall provide for monitoring execution of the procedures used in issuing and tracking the work tasking instructions for all work relating to the software development. Subjects to be monitored shall include, but not be limited to, the following: The description of the work to be accomplished; assignment of responsibility for its accomplishment; the manner in which the initiation of work is authorized and completion of work is certified; the manner in which periodic reports on the status of work against approved schedules and resources allocations are prepared and submitted; and scheduled completion dates.
- 10.2.3.2 Configuration Management (CM). Specify the quality assurance measures to be applied to CM. Describe methods for evaluation of the contractor's CM procedures to ensure that the objectives of the CM program are being attained. The plan shall designate responsibility for conducting the evaluations of CM and the expected number of frequency of evaluations. The Software QA Plan shall describe the method by which documentation of the evaluations is accomplished.
- 10.2.3.3 Software Specification, Design, and Coding. Describe all QA procedures to be applied by the contractor to specification, design, and coding of software.

The QA Plan should describe and distinguish between the procedures applied by the QA function and the procedures applied by the software developers that support, complement, or implement the QA objectives. The QA Plan shall describe how the results of the application of these procedures will be documented. QA procedures to be described in this area shall include, but not be limited to, the following:

1. Review of functional and performance specifications to determine conformance to and compliance with higher level requirements as expressed in operational requirements documentation, applicable military standards and military specifications, the statement of work, and other governing specifications contained in or invoked by the contract.
2. Review of all interface design, program design, data structure and data base design specifications and documents to evaluate the respective areas in terms of efficiency, effectiveness, reliability, testability, maintainability, and conformance with higher level specifications and standards. The QA Plan shall state whether or not the accomplishment of the design documentation review occurs prior to the release of the computer program design to coding.

3. Review of the coding process to ensure that the program is being coded in accordance with approved design specifications and applicable programming standards and conventions.
  4. Procedures by which allocated computer program resources and actual utilization are to be monitored and reviewed to ensure that established constraints for processing time, storage, and input/output channels are not exceeded, and the specified reserves are maintained.
  5. Procedures for applications of tools, techniques, and methodologies throughout the software development process to point out software defects as well as areas for potential improvement prior to formal testing.
  6. Procedures to be followed to ensure that the software development process is conducted in accordance with the approved software development plan.
  7. Procedures for certification of tools, techniques, and methodologies used throughout the software development process.
- 10.2.3.4 **Software Testing.** Describe all QA procedures relative to testing of the software. The specifying, defining, and scheduling of tests are contained in separate Test Plans and Test Specifications. The Test Section of the QA Plan addresses the procedures to assure the quality of all aspects of testing. Software Testing QA procedures to be described shall include but not be limited to those in MIL-S-52779 Paragraph 3.2.8.
- 10.2.3.5 **Corrective Action.** Delineate those contractor procedures which shall assure the prompt reporting and correction of deficiencies which have resulted in or could result in noncompliant software. Corrective action shall include as a minimum the requirements of MIL-S-52779 Paragraph 3.2.9.
- 10.2.4 **Documentation.** Describe the contractor's procedures for reviewing documentation for compliance with standards, conventions, and practices and compliance with contract requirements. The procedures shall identify how compliance and approval will be indicated on the documentation and the scheme for numbering draft and final versions of the documentation. The procedures shall address the distribution, update, and controls of the master and backup documentation. Where the contract is silent, the contractor shall identify documentation standards, practices, and conventions that will be used.
- 10.2.5 **Computer Program Library Controls.** Describe the procedures for the maintenance and control of the working library, the master library, and the contingency library. The plan must specifically describe the procedures for maintaining version accountability by limiting access to and by controlling internal (contractor) and contractual baselined code.

- 10.2.6 **Reviews and Audits.** Describe the procedures that the QA Function will use to ensure that the preparation and execution of formal reviews and audits (e.g., PDR, CDR, FCA) are conducted in accordance with the prescribed procedures and are accurate representations of the current software and documentation status. The role of the QA function in establishing traceability of initial contract requirements through successive baselines shall be included or referenced. Subcontractor review and audit schedules shall be described and synchronized with the prime contractor review and audit schedules. The QA Plan shall also describe any additional reviews and audits; both scheduled and unscheduled QA reviews and audits shall be described.
- 10.2.7 **Subcontractor Control.** Describe procedures and responsibilities for assuring that all software developed by subcontractors is developed in accordance with the requirements of MIL-S-52779 and any other contractual requirements. Describe the degree of prime contractor participation in subcontractor(s) software QA program(s). The review of subcontract statements of work, review, and audit of subcontract procedures and progress, and inspection and acceptance of subcontractor-developed deliverables shall be discussed.
- 10.2.8 **Final Certification.** Describe the procedures and responsibilities for the final certification that all delivered software products meet all contractual obligations, including both software and system performance objectives.
- 10.2.9 **Reporting and Control System.** Describe the reporting and control system that will be employed to:
1. Permit management to monitor overall quality status of the software and documentation
  2. Permit decisionmaking on the basis of quality assurance data
  3. Bring inadequacies, discrepancies, and deficiencies, as well as proposed improvements, to the attention of appropriate supervisory and management personnel in a timely manner
  4. Permit rapid and effective corrective action with positive feedback and response.
- 10.2.10 **Preparation and Delivery.** Describe the QA measures to be applied to assure the integrity of software products during handling, storage, preservation, packaging, and shipping.
- 10.2.11 **Plan Implementation.** Describe specific tasks, responsibilities, and resources necessary to implement the Software Quality Assurance Plan. Show personnel resources to be assigned to the QA function, including number and background of individual QA personnel, duration and percentage of time they will be assigned to the QA function, and duties they will be assigned (if any) while not performing QA.

#### **NOTES ON APPENDIX 4**

Dr. Schneidewind would like the following comments noted:

1. Paragraph 3.1 should state that the QA Plan should also apply to maintenance.
2. The first sentence of Paragraph 10.2.3.3 should include maintenance.

APPENDIX 5 - COMMENTS ON PANEL C REPORT

DEPARTMENT OF THE AIR FORCE  
Headquarters Rome Air Development Center (AFSC)  
Griffiss Air Force Base, New York 13441



Reply to  
Attn. of: ISIS, (315)330-4325

5 Jun 1979

SUBJECT: Comments on Panel C Report, JPCG-CRM Software Workshop

Lt. Col. H. Oberkrom  
HQ AFLC/LDEC  
Wright-Patterson AFB OH 45433

1. My comments will be consistent with the page numbers and the paragraph designation system of the May 18, 1979 report to all members of Panel C from Robert Dunn. In addition, I will concentrate in the sub-panel area defined as "DID for a software QA plan".

2. Comments:

(a) Page 7, Paragraph 4.1 - In the discussion of the matter of collecting error data I feel that there is really little or no cost penalty since some companies include software data collection as part of their Quality Assurance Plan. They do this for management control of the software development process and to provide data for research in software development methodology. It is possible that not all organizations specifically collect error data but highly improbable. The cost penalty could be incurred in the analysis of this data but analysis would not always be required.

(b) Page 9, Paragraph 4.3 - Specific type software developments (not necessarily weapons systems) could require analysis, thus a separate DID is in order or should be an option within the proposed change to DI-R-2174. At this time I disagree that there are any existing DIDs that fulfill this need. In going over the package of DIDs that were received prior to and at the Workshop, I find the implication for data collection but nothing that is specific enough. In the event I missed the existing DIDs that fulfill the need, please inform me where I may obtain them.

Regarding terminology, not only in the Quality Assurance area but in general in the Software Community, I feel progress has been slow in doing 'something' with MIL-STD-109B. Not having any results of the recommendations of the other panels, I would like to go on record as stating that this standard can be improved tremendously just from the work in recent years by the IEEE Software Engineering Standards

Subcommittee on Terminology. As a whole, the MIL-STD-109B is inadequate for most procurements....

*James V. Cellini, Jr.*

James V. Cellini, Jr.  
Software Sciences Section  
Information Processing Branch

Cy to: Mr. Robert H. Dunn  
Dept 63401  
ITT Avionics Division  
100 Kingsland Road  
Clifton NJ 07014

**PROCEEDINGS OF THE SOFTWARE WORKSHOP  
JOINT LOGISTICS COMMANDERS  
JOINT POLICY COORDINATING GROUP ON COMPUTER RESOURCE MANAGEMENT  
MONTEREY, CA, 2-5 APRIL 1979**

**Report of the Panel on  
SOFTWARE ACCEPTANCE CRITERIA**

**Chairman: R. Dean Hartwick  
Logicon, Inc.**

## OBJECTIVE

Develop recommendations for the Joint Logistics Commanders Joint Policy Coordinating Group (JLC-JPCG) on Computer Resource Management, as the first step toward a triservice standard that delineates criteria for the acceptance of embedded computer software.

## SCOPE

Currently, no definitive criteria for the acceptance of embedded computer software are provided through the existing military standards; as a result, the acquisition program manager often has no assurance that computer software, if accepted, will perform operationally at an acceptable level. Successfully supplying such criteria not only should satisfy the primary objective of delivering operational software that works, but also should:

1. Allow the extent of acceptable developmental progress to be quantitatively measured.
2. Improve visibility into the developmental status of software throughout the developmental cycle.
3. Provide a better basis for software acquisition managers and software developers to agree on job completion criteria.
4. Provide a basis to better express the quality of delivered software.

This panel was assigned to review military standards, regulations, and instructions that are pertinent to the acquisition and quality assurance of computer software. It was assumed that the appropriate application of software acceptance criteria could be developed as stand-alone documents, as modifications or interpretations of existing documents, or a combination of the two. The panel's objective was to determine the best way of implementing software acceptance criteria based upon how they might best work, require the least perturbation to existing documents, and be expeditiously implemented. The panel also considered the sequence of steps through which the panel findings could eventually be utilized.

## APPROACH

The panel on software acquisition acceptance criteria consisted of the 14 members named in Appendix 1. The panel had previously been informed through a letter from the panel chairman (Appendix 3) of the purpose and objectives of the workshop. After the general workshop meeting had adjourned, the panel met for an hour on April 2 to discuss the panel charter, and the procedure to be followed in achieving it. At this meeting, the panel received a handout suggesting the procedure for further dividing the issue into three smaller questions and appointing subpanels for detailed analysis (Appendix 4).

On the morning of April 3, the complete panel met and received presentations on viewpoints held by panel members from different services. Presentations were made by:

1. J. Gary Nelson
2. Richard Mitchell
3. Alton Patterson

U. S. Army Test and Evaluation Command  
Naval Air Development Center  
Sacramento Air Logistics Center

A copy of visual aids used by Mr. Nelson and Mr. Patterson and summaries of the talks of Mr. Mitchell and Mr. Patterson are contained in Appendix 5.

Following the presentations, the entire panel arrived at a number of consensus positions, which are summarized and discussed in the section of this report titled "Discussion." With these consensus points established, the panel formed the following subpanels for more detailed analysis and discussions:

<u>Standards Interface</u>	<u>Acceptance Procedures</u>	<u>Software Error Theory</u>
Stanley Brown	Robert Alger	Roger Bate
Brinton Cooper	Brian Butler	Gene Walters
Paul Reimann	Frank Phillips	Gary Nelson
George Tice	Richard Mitchell	Boyd Wilson
	Alton Patterson	

For the rest of the workshop, the panel alternated between subpanels considering their detailed areas, subpanels presenting findings to the panel, and broader panel discussions.

On the afternoon of the first day, the panel chairman gave a short presentation of panel objectives to the entire workshop. Thereafter the panel results were summarized each day by the panel chairman. Visual aids used in these presentations are contained in Appendix 6.

The panel reached a consensus that a final meeting was required to complete its report. Included in this report would be two draft documents for submission to the JLC-JPCGCRM. These are contained herein as Appendices 7 and 8, and are discussed in the next section, Discussion. Accordingly, a smaller group of the panel met in San Pedro, CA, on May 22, 23, and 24 to finish the draft report. The following panel members participated at this session:

Robert Alger  
Roger Bate  
Marilyn Fujii  
R. D. Hartwick  
Richard Mitchell  
Alton Patterson  
Paul Reimann

Throughout all panel meetings, Paul Reimann served as secretary. His excellent notes were essential in drafting this report.

## DISCUSSION

The first issue addressed by the panel was:

"What is the problem that developing a standard set of acceptance criteria solves?"

The problem addressed was articulated as follows:

Problem: A need exists to measure embedded computer software performance to ensure that the software will perform acceptably when used.

The two words that are underlined presented the two significant problems faced by the panel. First, it wished to deal as much as possible with specifics that could be understood and applied by a variety of people throughout the triservices. It could not be assumed that experienced software engineers would be the only people applying the criteria. Therefore, the criteria had to be meaningful and, if possible, quantifiable to ensure uniformity and fairness of application. Second, the criteria had to measure the appropriateness of the software to accomplish its intended mission. The underlying purpose of this entire effort is to ensure that both operational and support software, when delivered to be used, will work through its life cycle.

Additional advantages should result from the application of a good set of acceptance criteria. These include:

1. Because it ensures that requirements are codified early, program offices are able to detect and correct software problems at the best possible (cost effective) time.
2. A better basis is provided to record and communicate software status.
3. A negotiated basis is provided for developers and users (or contractor and buyers) to ascertain completion.
4. Additional support for program review is provided.
5. Management is given enhanced visibility into software development progress.

The presentations by panel members and resulting discussions revealed several consensus points on how all of the services and industries represented actually improve software acceptance criteria in their ongoing acquisitions of new software and maintenance of existing software.

#### 1. Software Acceptance Criteria Should Be Applied Throughout The Acquisition Cycle.

Based on their collective experience, panel participants unanimously rejected the notion that an adequate job can be done by running acceptance on a "black box" testing basis. Every testing scheme discussed emphasized involvement throughout the acquisition. Special emphasis was accorded to ascertaining that initial software requirements are properly stated. Thereafter, there is a continual checking against increments of software development.

Design must completely and properly embrace all requirements. Code must follow design specifications. Tests should be conducted to plan and be responsive to demonstration of requirements and specifications. At the completion of these activities, there is a place for a final demonstration of the software that is akin to "black box" testing. This final testing represents only a small amount of the total acceptance process.

Because of this consensus, the panel was unable to agree on TADSTAND 9 as the baseline acceptance document, because it primarily addressed a narrow time within the overall acquisition period. Further, the quantitative values given for acceptance/rejection were considered dangerously simple when weighed against all the nuances of software. Applied in the hands of an unknowledgeable software engineer, they could be used to reject substantially all software ever developed.

On both a practical basis and technical basis, the panel therefore determined that acceptance must be provided throughout the acquisition cycle and enhancement cycle of operational software. As a result the following conclusion was reached:

Conclusion 1: Software acceptance criteria must be applied incrementally at meaningful events throughout the software life cycle.

2. Software Acceptance Criteria Are Most Critical At The Time of Developing Requirements and Design Specifications.

The panel unanimously concurred that the most critical stage to address the acceptability of software products was in the earlier stages of development. If the proper requirements are not adequately stated at commencement of a software development, the whole effort will be jeopardized in time, effort, cost, and technical quality. All present stressed the need to apply meaningful criteria at these vital early stages. This not only enforces the previously stated Conclusion 1, but also points out a problem that many organizations have. Frequently, at this most important stage of the development process, the managers are in a learning stage and are unable to measure the acceptability of software deliverables.

Further, this early measurement sets the base for the continuing process of evaluating software thereafter. Each step thereafter is essentially tracing and measuring how well a subsequent translation implements the earlier specification. Thus it is critical to ascertain acceptability at the outset.

Based upon these factors, the following conclusion was reached:

Conclusion 2: The software acceptance criteria should stress the means of ensuring the acceptability of requirements and the subsequent tracing of requirements throughout the acquisition.

3. Software Consists of Program Code, Data, and Documentation.

It was agreed that a definition of software was necessary in order to focus on software acceptance criteria. The panel was working with embedded computer software. This might be envisioned as a part of an information system component of a larger (undefined) system, shown in Figure 1.

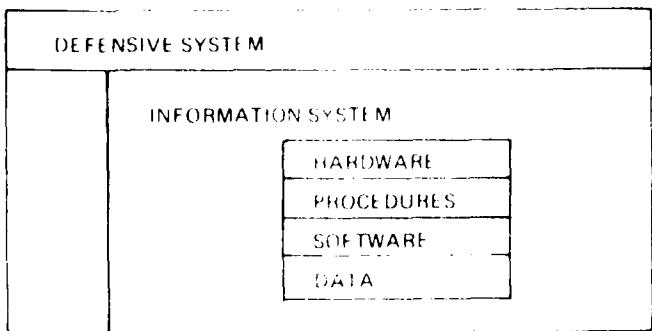


Figure 1. Sample System With Embedded Software

Within this framework, it was agreed that, for purposes of applying acceptance criteria, operational and support software consisted of:

- a. All code, including microcode (or firmware)
- b. All documentation that specifies this code (e.g., requirements, specifications), and is required to test and use it (test plans, user manuals).
- c. Any data resident with the code (e.g., constants, parameters, initial test data set).

The point concerning the data set can be elaborated. The data management software is considered to be a part of items a and b. The data handling should therefore be checked out as a program testing function. Procedures to qualify data should be included as documentation. The actual steps of qualifying operational data constitute a procedural question that is a part of using the software. The first data that is used to accept software does need to be accepted and is included under item c.

Conclusion 3: Software is defined as computer code, constants and parameters, documentation, and initial data set.

#### 4. The Agency Applying Software Acceptance Criteria Should Be Independent of the Developing Agency.

The panel endorsed the concept behind independent test and evaluation (or independent verification and validation), that the ensurance of acceptable software is best performed by an agency independent of the developer. The value of this method stems from the true objectivity that can be derived by an independent group. The means of achieving independence (independent contractor, independent agency, test group, etc.) is an organizational question, and can be solved in different ways. It is most important that the group be free of the bias that comes from earlier participation and retain that objectivity through careful control of interfaces with the developer. Although this aspect of independence was considered very important by the panel, it was considered to be beyond the scope of the panel charter to fix its usage as a matter of policy. Therefore, the panel agreed that desirability of independence would be stated as a conclusion, but that conclusion would be picked up as an action recommendation. The question is suggested as a topic for additional analysis by the acquisition panel (Panel A).

Conclusion 4. Software acceptance criteria are best applied by a test and evaluation (verification and validation) agency that is independent of the developing agency.

5. Software Must Be Accepted At Different Levels And Software Acceptance Criteria Should Be Addressed To All Of These Levels.

A common difficulty of developing software for embedded computer systems is that software typically cannot be developed or tested for one level of operation. It is quite common to have a nested hierarchical dependence of information system components. Software can be embedded within software, embedded within subsystems, and embedded within an overall defense system. One can never be sure that software performs correctly or adequately until the entire system has been integrated and certified. (This is another reason why the panel reached Conclusion 1 - it is virtually impossible to state categorically that, at any one test point, the software can be totally accepted and considered to be error free). The panel envisioned an evolutionary acceptance pattern that commences at the time system requirements are generated, propagates throughout software acquisition, and finishes at completion of operational test and evaluation. Once operational, much the same cycle of events is required in a modified form for enhancements and maintenance of operational software.

The panel determined that it should not attempt to relate this conclusion to contractual matters for two reasons. First, the conclusion should be equally as applicable to a development performed solely by the Government, where no contracts are involved. Second, it was felt that the panel should say what should be done to accept software. The tradeoffs on how this can best be done should be left to the discretion of program offices based upon their unique applications, constraints, and requirements. Therefore, the panel has elected to say that software acceptance cannot be completed until the embedded system has been completed. However, criteria are applied about when it is prepared to make a contractual acceptance based upon the extent of the technical acceptance to that point.

Conclusion 5: Software acceptance is finally achieved only when the embedded computer system has been accepted.

6. Software Acceptance Criteria Can Be Applied Now Using Existing Acquisition Standards.

A subpanel addressed the issue of whether existing acquisition standards can be used to apply software acceptance criteria. If so, how and why are modifications required? If not, how should a new standard interface with the existing standard structure? The subpanel determined, and the full panel accepted their finding, that:

- a. No new standard is required or even advisable.
- b. No changes to existing standards are required to implement the application of software acceptance criteria now. (A number of the subpanel did develop a considerable number of suggested changes that were submitted independently, without panel consideration.)

- c. A guide is required for application of acceptance criteria within the existing standards framework.

Table 1 shows the array of applicable DOD standards that were considered by the subpanel. The panel also considered MIL-STD-1679, which was not broken out on the figure. The subpanel determined that an ideal acquisition cycle had to be developed that could be used as a standard for examining the various documents enumerated in Table 1.

Accordingly, an ideal four-phase embedded computer acquisition cycle was developed. Within each phase, the applicable standards were summarized in terms of approval/disapproval events within four categories:

- a. Reviews and Audits
- b. Documentation
- c. Configuration Management
- d. Test and Evaluation.

The results of this effort are provided in Table 2.

Based upon this information classification, the various standards were examined to identify where the provision for the ideal approval/disapproval event is made. Table 3 summarizes where these identities were made for the principal software acquisition standards. Those sections must be correlated to a guidebook wherein the software manager can be instructed on what acceptance criteria might be applied. As a result of this analysis, the panel reached the following conclusion:

Conclusion 6: The existing standards, directives, and regulations pertaining to software acquisition do not need to be modified to apply software criteria, but a guidebook showing how to relate software acceptance criteria to them should be written.

#### 7. Software Acceptance Criteria Should Take The Form Of A Series Of Checklist Items Applied At Approval/Disapproval Events With The Acquisition Cycle.

A subpanel addressed the very complicated issue of the extent to which a software error theory existed or could be codified for triservice applications. The panel had previously considered the acceptance standards of TADSTAND 9 to be inappropriate in that they applied at only one cyclical event and were totally "black box" oriented. In addition, it was felt that the pass/fail criteria were too arbitrary in nature, giving rise to the danger of, on one hand, being inadequate to prevent bad software from going operational or, on the other hand, needlessly holding up fielding a system that would be adequate.

Table 1. DOD Software Standards

APPLIES GENERALLY NOT SPECIFICALLY TO SOFTWARE, OR IS AMBIGUOUS WITH RESPECT TO SOFTWARE FIGURE OBTAINED FROM UGRAPH USED IN KEYNOTE BY BRUGADER GENERAL LASHER

Table 2. Ideal Software Acquisition Cycle

Functional	Baseline	Allocated	Baseline	Product Baseline	Operational	Baseline
Systems Requirements	Specification and Design		Development			Support
Development	Development		Development			Maintenance
Yes	No ◇ Yes	No ◇ Yes	No ◇ Yes	Reviews and Audits (approve/disapprove)	Reviews and Audits (approve/disapprove)	No ◇ Yes
Reviews and Audits (approve/disapprove)	System Requirements (SRR)	Systems Design (SDR) Preliminary Design (PDR)	Critical Design (CDR) Functional Configuration Audit (FCA)	Physical Configuration Audit (PCA)	Reviews and Audits (approve/disapprove)	Formal Qualification (FQR)
Documentation (approve/disapprove)	System Specification Software Development Plan Software Quality Assurance Plan Project Management Plan Integrated Logistics Plan Support Software Plan	Program Specification Program Design Specification Interface Design Specification Data Base Document	Program Design Documents Program Package	Documentation (approve/disapprove)	Documentation (approve/disapprove)	Configuration Management (approve/disapprove)
Configuration Management (approve/disapprove)	CM/CSA Plan	CM/CSA Software Errors Software Patches	CM/CSA Software Errors Software Patches	CM/CSA Software Errors Software Patches	CM/CSA Software Errors Software Patches	CM/CSA Software Errors Software Patches
Test and Evaluation (approve/disapprove)	Test and Evaluation Plan	Subprogram Tests Function Tests	Performance Tests	Test and Evaluation (approve/disapprove)	Test and Evaluation (approve/disapprove)	Test and Evaluation (approve/disapprove)
					Software Integration Test System Integration Test Acceptance Tests Technical/Operational Evaluation	Software Integration Test System Integration Test Acceptance Tests Technical/Operational Evaluation

**Table 3. Correlation of Standards and Ideal Cycle (1 of 4)**

**A. System Requirements Phase Sections**

<u>Reviews/Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<b>Revised</b>
Sys Rqmts (SCR)	App. A				
Sys Design (SDR)	App. B				
Prel Design (PDR)	App. C				
<b><u>Documentation</u></b>					
Sys Spec	App. A,B	App I, 10	App. II, 30		
S/W Rev Plan					
S/W QA Plan				5.9	3.1
Prog Mgt Plan					
Integ Log Plan					
Support S/W Plan					
<b><u>Configuration Management</u></b>					
CM/CSA Plan					
<b><u>Test and Evaluation</u></b>					
Test and Evaluation Plan	10.4				

**Table 3. Correlation of Standards and Ideal Cycle (2 of 4)**

**B. Specification and Design Phase Sections**

<u>Reviews/Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<u>Revised 52779</u>	<u>480</u>
Sys Design (SDR)	20.				3.2.6	
Prel Design (PDR)	30.				3.2.6	
<b><u>Documentation</u></b>						
Prog Spec	20.	60.	60.4	5.1	3.2.2, 3.2.4, 3.2.8	
Prog Design Spec	40.	130.	60.5	5.2	3.2.2, 3.2.4	
Interface Design Spec	10.		20.		5.12.4b, 5.12.5c, 5.2.3	
Data Base Document	30.				5.2.2.6	
<b><u>Configuration Management</u></b>						
CM/CSA			140.	5.11	3.2.7	
Software Errors				5.10.3	3.2.9	
Software Patches				5.10.3.2		
<b><u>Test and Evaluation</u></b>						
Subprogram Tests		30.4		5.10.2	3.2.8	
Function Tests		60.4			3.2.8	

**Table 3. Correlation of Standards and Ideal Cycle (3 of 4)**

<u>C. Development Phase Sections</u>						
	<u>Reviews/Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<u>Revised</u> <u>52779</u> <u>480</u>
<b>Critical Design (CDR)</b>	3.4 40				4.4 5.12.3	3.2.6
<b>Function Configuration Audit (FCA)</b>	3.5 50				4.4 5.12.3	3.2.6
<b>Physical Configuration Audit (PCA)</b>	3.6 60				4.4 5.12.3	3.2.6
<b>Documentation</b>	60.1					
<b>Program Design (C5=Part II, etc)</b>	40.1.3.1(a) 40.1.3.2(a)	130		60.1,60.2 60.2.1(b) 60.5	3.6.2,5.2 5.12.3	3.2
<b>Program Packages (program on deliverable media, in source and object, plus listings, X-refs, etc.)</b>	40.1.3.2	4.5		60.5.5 60.5.4.2	5.5.6 5.5.7, 5.6, 5.7 5.12.3.1(d)	5.0
<b>Configuration Management</b>						
<b>CM/CSA</b>		3.1.1		Entire document	4.5,5.11	3.2.7
<b>Software Errors</b>					3.9.1, 5.8.5, 5.10, especially 5.10.3.1 3.15,5.10,	
<b>Software Patches</b>					3.2	
<b>Test and Evaluation</b>						
<b>Performance Tests</b>	3.5 50	60.4		60.4.4 60.5.4	5.8.3, 5.10	3.2.8

**Table 3. Correlation of Standards and Ideal Cycle (4 of 4)**

**D. Evaluation Phase Sections**  
**After Product Baseline - Before Operational Baseline**

<u>Reviews/Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<b>Revised</b>
<b>Formal Qualification Review (FQR)</b>					<u>52779</u> <u>480</u>
	App. G	4.3.2 4.4 App. VI App. XIII		App. VI 5.11.1.2	3.2.8
<b>Documentation</b>					
Operator's Manuals	App. G	App. VI	App. VI	4.4	
User's Manuals		60.5.4.1	60.4.4	5.8	3.2.4
Test Reports		App. XIII	60.5.4.1	5.9	3.2.8
<b>Configuration Management</b>					
CM/CSA					
Software Errors					Para
Software Patches					9.1
<b>Test &amp; Evaluation</b>					
Integration Test					
System Integration Test		4.4.1.2	4.4.1.2		
		4.4.2	4.4.2		
Acceptance Test		4.6.4	4.6.4		
Tech. Eval. Op. Eval.		App. VI	App. VI		

The subpanel defined general categories for classifying software errors within specifications, code and data. These are provided in Table 4. (The subpanel considered this a preliminary and therefore incomplete enumeration.) It was felt highly desirable to develop a statistical data base that could be used to provide direction to software managers. For example, the subpanel believes that historical data could be used to indicate when testing has reached a maturity level that permits the next following test phase to be commenced; an example of how this could work is shown in Figure 2. Here, a profile of errors detected is plotted as a function of testing phase and time. The panel speculated that a quantifiable algorithm could be specified as a function of present error count against peak error count that identifies a sufficient level of maturity to release to the following testing phase. Once a sufficient data base of software errors exists, then a software error theory for different generic classes of software could and should be developed.

Within the constraints of present knowledge and statistics, the panel concluded that the software acceptance criteria should be applied as a subjective checklist at each approval/disapproval event. But it further felt that the JLC should sponsor the effort to collect error statistics from embedded computer software developments and derive a working error model with this base.

Conclusion 7: Software acceptance criteria should be developed as a checklist of subjective criteria for imposition at all approval/disapproval events within the software acquisition cycle.

Conclusion 8: The JLC should sponsor the collection of software error statistics on all embedded computer software developments in order to build a statistical base, and develop a software error model using this base.

#### 8. The JLC Should Mandate the Use of Software Acceptance Criteria and Prepare a Guidebook for Software Managers to Use.

One subpanel attempted to define procedures for applying software acceptance criteria. These procedures in effect amounted to the checklisting of criteria by acquisition cycle event.

In discussions with the entire panel, it was determined that event procedures could be correlated to the standards acceptance correlation (Table 2) to serve as an interim guide for a software manager to apply software acceptance criteria. A concern existed with the time dependency or criticality: that less exists to be evaluated at PDR than at CDR, and so on.

A criteria-event matrix to reconcile concerns of acceptance criteria relating to acquisition milestones and concerns of acceptance criteria relating to the particular product was proposed. A list of acceptance criteria would be drawn up (one list for each product). Then each criterion would be correlated to the milestone(s) concerned with checking that criterion. A scale of involvement, running from 0 (minimal) to 3 (heaviest involvement) for each milestone, can then be applied.

Thus, a user could observe the intersections of the product acceptance criteria with the particular successive milestones to define what criteria will be scrutinized (and at which depth) at each successive milestone for that product.

**Table 4. Software Error Categories**

**Software Specifications**

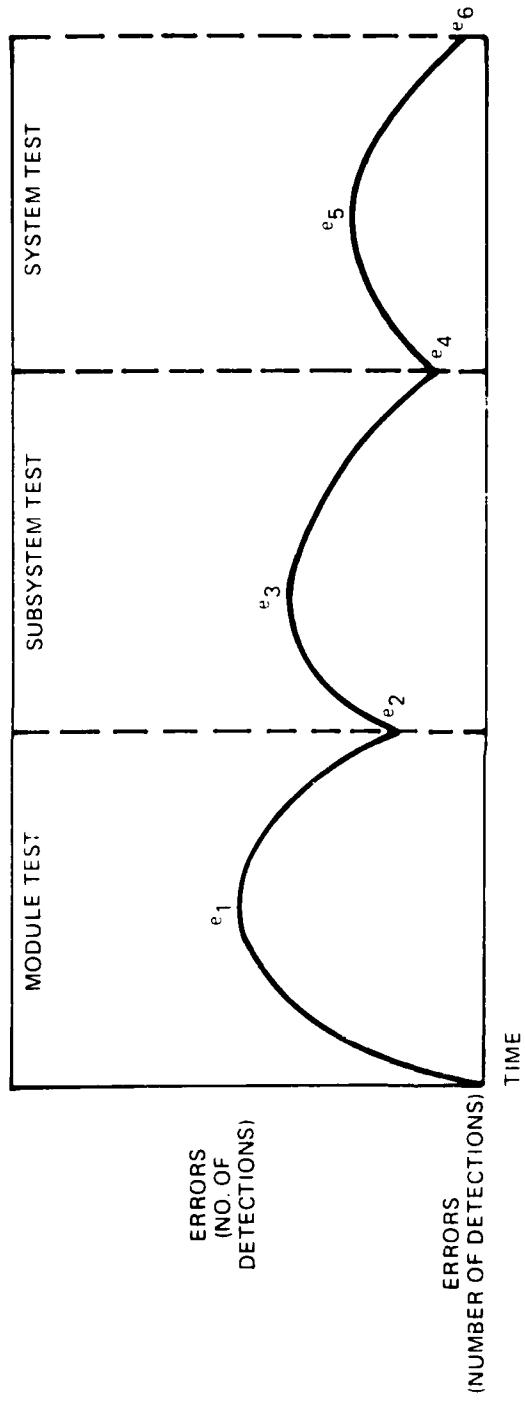
1. Unnecessary functions
2. Incomplete requirements or design
3. Inconsistent requirements or design
4. Untestable requirements or design
5. Requirements not traceable to higher specifications
6. Incorrect algorithm
7. Incomplete or inaccurate interface specifications

**Code**

1. Syntax errors
2. Noncompliance with specification(s)
3. Interface errors
4. Exception handling errors
5. Shared variable accessing error
6. Software support environment errors
7. Violation or programming standards
8. Operational support environment errors

**Data**

1. Accuracy
2. Precision
3. Consistency



MODULE TEST COMPLETE WHEN  $e_2 = p_1 e_1$   
 SUBSYSTEM TEST COMPLETE WHEN  $e_4 = p_2 e_3$   
 SYSTEM TEST COMPLETE WHEN  $e_6 = p_3 e_5$   
 WHERE  $p_1$  IS A MODEL PARAMETER USED TO DETERMINE  
 WHEN THE NEXT PHASE CAN BE ENTERED.

Figure 2. Error Model

It was agreed that the guidelines should be for "novices", and should provide a common understanding of what is required from all participating parties: Program Manager (PM), contractor, user/maintainer, etc.

It is necessary to establish a list of the applicable products for the acceptance procedures and acceptance criteria. A list of possible candidates came from Figure 1 of Panel A's Final Report on Software Acquisition and Development. Of the documents listed there, the following were chosen. (The system level spec set was not on that chart, but was included in the candidates for acceptance procedures and criteria because it is a source for later or succeeding documents which require such acceptance criteria definitions.) The selected sets were:

- a. The set of "system level specs" which comprise the Functional Configuration Identification at the time of the Functional Baseline at DSARC I, though this FCI set is updated as necessary after the Functional Baseline
- b. The Development (Part I or B 5) Spec or Performance Requirements Spec
- c. The Product (Part II or C 5) Spec or Product Design Spec
- d. The CPCI product (code)
- e. The CPCI Test Plans
- f. The CPCI Test Procedures
- g. The CPCI Test Reports
- h. The Handbooks and Manuals (User's or Operator's Handbooks, etc).

It was decided to exclude specific acceptance criteria for the CM, DM, QA, and CP Development Plans which appeared on Panel A's Figure 1. Though they are constraints on the development of the product CPCIs, they are not specifically part of the product. Also, these items could be added at a later date. The same exclusion was decided upon for the CM status reports and change processing forms listed on Panel A's Figure 1. The VDD shown on Figure 1 was considered to be part of the CM package, not as a product in the strict sense, so it too was excluded from consideration as a product for which acceptance procedures and acceptance criteria would be developed.

For these criteria to have real significance in a program, the software manager should be provided a suggested "action" to take for failures. A truly useful action list requires a three dimensional perspective. The three dimensions are failure severity, total cost of correcting the error, and cost schedule slope (rate of extra cost or adverse schedule slippages the longer the error remained unfixed). Classifications of priorities of correcting errors were developed as "emergency, urgent, and routine", interpreted as functions of cost and schedule slippages. Specifically:

- a. A failure requiring an "emergency" fix was a failure which, if not fixed, imposed rapid increases in cost (possibly including safety considerations) or caused ever greater schedule slippages.
- b. A failure requiring an "urgent" fix was one which could cause moderate increases in cost or moderate but perpetually increasing schedule impacts (delays) the longer it remained uncorrected.
- c. A failure classified as "routine" was one which caused no particular cost or schedule slippage no matter how long it remained uncorrected.

This action classification was considered to be a "nonmandatory" guide of how decisionmaking might be undertaken for determining how (if at all) to fix different kinds of software errors, depending upon the consequences of not correcting the errors as the program progressed, or of not correcting them immediately. Different systems might respond uniquely.

Originally, the panel intended to use the 5 error classifications found in MIL-STD-1679 and TADSTAND 9 as the error types. But though these are not specifically limited to failures found during testing, they are defined in MIL-STD-1679 in the section on testing and seem to refer only to errors in the CPC1 software itself, such as might be found during acceptance testing. Since it had been previously agreed that software acceptance criteria would look at all aspects related to the embedded computer system software, including documentation and support software, the subpanel decided to define some failure categories that could also include support software and documentation. These were boiled down to four failure severity categories (including operation software and documentation):

- a. Severity 1: "Prevents accomplishment of its primary function, jeopardizes safety, or inhibits maintainability of the software."
- b. Severity 2: "Degrades performance or maintainability, with no workaround."
- c. Severity 3: "Degrades performance or maintainability, but a workaround exists."
- d. Severity 4: "Doesn't adversely affect performance or maintainability" (for instance, documentation errors transparent to users).

The panel determined that the most important output of applying acceptance criteria is preparing a guide for the software manager to take acceptable action where violations of acceptance criteria are encountered. In a three dimensional matrix, the following "actions" were defined to be required, depending upon the combination of failure severity, total cost, and slope (rate of change) of eventual cost and accumulated schedule delays:

- a. 'R' = "Rework it right now" (fix it immediately, even if this requires that other development or progress be halted for the moment).

- b. 'C' = "Correct the failure while continuing the development activity, etc."
- c. 'L' = "Proceed with no action" ("live" with the failure, and with a note to correct later if possible).

For purposes of determining failure severities, a failure or deficiency could be either 'singular' or 'plural'. That is, a large aggregate of similar small errors might have the effect of a single large deficiency, in that such an accumulation of "small" failures in some cases might have a severe impact if not corrected. The failure severity was thus taken to be equivalent to its impact singly or as a related group of failures, if the latter would have a more severe impact that must unavoidably be dealt with.

The figure consists of three vertically aligned tables, each with a corresponding label on the left. The top table is labeled 'HIGH COST/ SCHEDULE IMPACT', the middle one 'MEDIUM COST/ SCHEDULE IMPACT', and the bottom one 'MEDIUM COST/ SCHEDULE IMPACT'. Each table has three columns representing 'HIGH TOTAL COST', 'MEDIUM TOTAL COST', and 'LOW TOTAL COST'. The first two columns are grouped by a brace under the top table, and the last two columns are grouped by a brace under the middle table. The bottom table has no grouping brace.

HIGH COST/ SCHEDULE IMPACT			
HIGH TOTAL COST	MEDIUM TOTAL COST	LOW TOTAL COST	
R	R	R	SEVERITY 1 ERROR
R	R	R	SEVERITY 2 ERROR
L	L	P	SEVERITY 3 ERROR
L	L	R	SEVERITY 4 ERROR

MEDIUM COST/ SCHEDULE IMPACT			
HIGH COST	MEDIUM COST	LOW COST	
C	C	R	SEVERITY 1 ERROR
C	C	R	SEVERITY 2 ERROR
L	C	R	SEVERITY 3 ERROR
L	L	C	SEVERITY 4 ERROR

MEDIUM COST/ SCHEDULE IMPACT			
HIGH COST	MEDIUM COST	LOW COST	
C	C	C	SEVERITY 1 ERROR
C	C	C	SEVERITY 2 ERROR
L	C	C	SEVERITY 3 ERROR
L	L	C	SEVERITY 4 ERROR

Figure 3. Error Severity/Cost Matrices.

AD-A103 485

AIR FORCE LOGISTICS COMMAND WRIGHT-PATTERSON AFB OH  
PROCEEDINGS OF THE JOINT LOGISTICS COMMANDERS JOINT POLICY COOR--ETC(U)

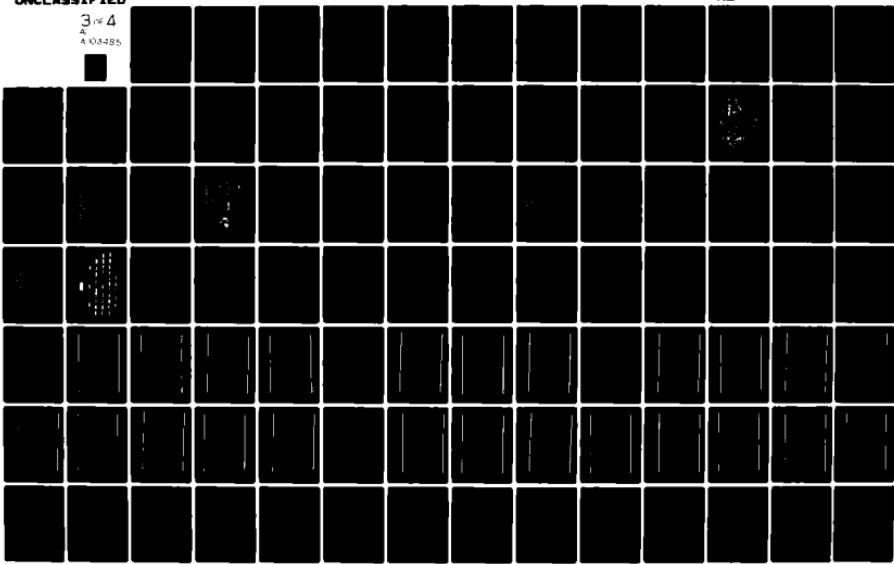
F/B 5/1

AUG 79

NL

UNCLASSIFIED

3 x 4  
A 103485



A high slope of schedule impact meant that for whatever period of time elapsed before fixing the failure, the cost per day was very high for not fixing it, or the schedule slipped badly each day that the failure was not yet fixed. The total cost included estimated costs for "average" delays in correcting the failure, as well as one-time costs unrelated to how quickly or slowly the error was corrected.

Finally, acceptance procedures need to be defined for each of the eight product types for which acceptance criteria are to be defined later. Acceptance procedures were distinguished from acceptance criteria as follows. Acceptance procedures were taken to be the types of actions which might be employed, such as different types of I/O or exception analysis, to help determine whether or not the product satisfied the requirements. By contrast, the acceptance criteria amounted to the presence and adequacy of various quality factors, such as adherence to various specified sound programming practices, or to presence and quality of input descriptions, etc.

The "strawman" starting point for developing the rough draft of acceptance procedures for each of the 8 products, was pages 95-100 of Volume II (Software Acquisition Process) of the Management Guide to Avionics Software Acquisition (ASD TR 76 11). Pages 95-100 deal with software verification and validation (V&V), but contain lists of V&V activities which could be applied. A separate list was given for each of the following areas listed in the document:

a. Requirements Analysis

This was taken to represent both the system level specs (FCI set), and the development (Part I or B 5) spec of performance requirements. To the list of V&V techniques for Requirements Specs were added the techniques of "Document review" and "Develop Traceability Mappings into successive documents". The original list included the following for requirements analysis: independent derivation of software requirements from system requirements; comparison to standard reference systems or similar systems, previously developed; functional simulations and modeling of process allocations; timing and sizing analysis, and the establishment of budgets for critical system parameters; and development of a requirements chart which identifies interrelationships between requirements.

b. Design Analysis

This was taken to represent the product spec (Part II or C 5, including data base spec) or program design spec. Only the category of "document review" was added to the analysis techniques listed in the Guide section. The entries included: correlation and traceability between design elements and software requirements; functional simulation to assess design integrity and process allocation; independent derivation of equations and algorithms; comparison with standard references and models; comparison with methods which have been proven in operational systems; mathematical and logical analysis.

c. Code Analysis of the CPCI itself

Three categories of V&V techniques were added: "interface analysis", which meant finding all calling hierarchies; tracing of I/O variables throughout the program, which meant taking a critical variable and "chasing" it through every occurrence of that variable in the program to be sure of proper initialization, scaling, and use, etc; and exception analysis, which meant checking error detection and interrupt handling and queueing, etc. These were added to the original list, which included the following: version comparison; text editing and syntax analysis; standards auditing; equation reconstruction from the code; data structure analysis; flowcharting and logic reconstruction; and manual code inspection.

The "Testing" phase in the report would correspond to the three separate categories of test plans, test procedures, and test reports. The paragraph did not break these out separately by document, but gave the broad sense given below: module testing (verify that individual software functions satisfy the corresponding software requirements); interface testing (verify that software-to-software and hardware-to-software interface functions are properly implemented); and system testing (verify that the operational system possesses the required system capabilities and satisfies the appropriate performance requirements). The panel then added these methods of analysis, applicable both to test plans and to test procedures: documentation review; requirements traceability analysis; coverage analysis, meaning how well test cases cover the possible input space, including both nominal and extreme values to exercise as much as possible of the code; and resource analysis, meaning analysis of the test bed resources and proper use of these in the tests.

The following techniques of acceptance procedures were added for the separate category of test reports: independent data reduction and analysis; independent review of test cases; and document review.

Finally, acceptance procedures or techniques were listed for handbooks and manuals, meaning user's or operator's manuals, as follows:

- a. Document review
- b. Mapping traceability to operator interface portions of the development (performance requirements or B 5/Part II), and product specs (Part II/C 5) or program design spec.
- c. Completeness analysis (have all possible contingencies been considered and taken care of, and have all functions been provided for?).
- d. Tests using operators.

A considerable effort was occupied in trying to find a common definition or terminology for "system level specifications" (the set of system specs and possibly segment system or subsystem specs which together comprise a Functional Configuration Identification (FCI). The FCI name was settled on, as identified in MIL-STD-480, with parenthetical explanations of some of the other names it may go by.

The FCI (system level set of specs) was the first of the eight products for which a rough attempt at defining acceptance criteria was begun.

This provided a model for all eight sets of acceptance criteria, as it was decided to list acceptance criteria one by one on the left of a table, with the previously mentioned intersections of criteria with particular milestones on the right. In order to show the relative importance of the various criteria in the different milestones, an importance scale was deemed necessary, ranging from 0 (least prominent in that milestone) to 3 (most prominent in that milestone):

- a. 3 = "primary approval cycle" (great emphasis on this criterion at this point)
- b. 2 = "the product is the basis for activity, such as traceability, but not the focus of primary emphasis"
- c. 1 = "only changes in its status will be looked at"
- d. 0 = "status check only (configuration accounting only; no actions)".

Using this criteria/milestone intersection code, the following table was developed for the FCI (system level spec set) checklist of acceptance criteria:

System level spec set (FCI) criteria	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Are s/w functions adequately defined for the system?	3	3	2	2	1	1	2
2. Is a test requirement adequately defined for the system?	3	3	2	2	1	1	2
3. Are s/w functional requirements consistent with user requirements?	3	3	2	2	1	1	2
4. Is the FCI in agreement with applicable documentation standards?	3	3	2	2	1	1	2
5. Are s/w functional requirements consistent with interfaces?	3	3	2	2	1	1	2

The panel unanimously concurred that an interim guide should be drafted and issued immediately for use now by software managers. Accordingly, the panel has prepared as Appendix 8 a summary of the material from which this guide can be prepared. The panel urges the JLC to prepare the interim guide from this material and issue it.

Additionally, the panel believed that these materials were already of sufficient value and should be applied to all ongoing developments. Accordingly, they believed that the use of the interim software acceptance criteria guidebook was important enough that the JLC should mandate its use. The panel has prepared a draft of such a mandate for JLC consideration as Appendix 7.

Finally, the panel addressed the question of the long term problems associated with applying software acceptance criteria. It was agreed that, while the interim guide should be prepared and issued quickly, it reflects the results of a short time effort and will not be a completely satisfactory guide. The panel agreed that the JLC, concurrent with issuing the interim guidebook, should start preparation of a final version of a triservice guide. The panel believed that a single unified set of software acquisition standards should be employed by all services. It was agreed that a final guide should be prepared to operate for both the final, unified standards and for the prior standards.

Conclusion 9: The JLC should mandate the use of software acceptance criteria for all software developments.

Conclusion 10: The JLC should issue a software acceptance criteria guide for use by software managers.

Conclusion 11: The JLC should continue work to develop a common, unified triservice acquisition standard.

Conclusion 12: The JLC should develop a final software acceptance guidebook to supplement existing acquisition standards and any future unified standard.

#### RECOMMENDATIONS

1. The panel recommends that the JLC adopt a triservice policy that mandates the use of software acceptance criteria incrementally throughout the software life cycle. On a short-term basis, this mandate could use the draft material contained in Appendix 7 as its basis.
2. The JLC should develop an interim guidebook to software managers that gives them the insight to implement the software acceptance criteria mandate through application of existing regulations and standards. The panel has prepared draft materials that can be used as the basis for this guidebook (Appendix 8).
3. The JLC should continue work toward developing and using common embedded computer software acquisition standards for all services. These unified standards should explicitly include software acceptance criteria.
4. A final triservice guidebook should be developed based upon the interim guidebook (recommendation 2) that gives practical working level knowledge of applying software acceptance criteria. This guidebook should be developed for existing standards and for any triservice standard.

5. The JLC should require that a record of software acceptance criteria failures be maintained so as to provide a statistical error base for improvements to the process.
6. The JLC should sponsor the development of software error models of generic classes of software.

APPENDIX 1 - PARTICIPANTS

APPENDIX 1 - PARTICIPANTS

**CHAIRMAN:** Mr. R. DEAN HARTWICK (213) 831-0611

Logicon, Inc.  
Strategic and Information Systems Division  
ATTN: MR. R. DEAN HARTWICK  
255 West Fifth Street  
P.O. Box 471  
San Pedro, CA 90733

Commander (202) 274-9651/2  
U.S. Army Material Development and Readiness AV 284-9651/2  
Command  
ATTN: DRDCE-RR  
(MR. BRIAN BUTLER)  
5001 Eisenhower Avenue  
Alexandria, VA 22333

Director AV 283-4030/2366  
U.S. Army Material Systems Analysis Activity  
ATTN: DRXSY-CC  
(DR. A. BRINTON COOPER, III)  
Aberdeen Proving Ground, MD 21005

Commander (301) 278-2775  
U.S. Army Test and Evaluation Command AV 283-2775  
ATTN: DRSTE-AD-S  
(MR. J. GARY NELSON)  
Aberdeen Proving Ground, MD 21005

Commander AV 441-3176  
Naval Air Development Center (215) 441-3176  
ATTN: Code 504  
(MR. RICHARD MITCHELL)  
Warminster, PA 18974

MR. FRANK A. PHILLIPS (301) 863-4787  
Systems Engineering AV 356-4787  
Naval Air Test Center  
ATTN: Code SY-43  
Patuxent River, MD 20670

Commander (714) 225-7015  
Naval Ocean Systems Center  
ATTN: Code 9133  
(MR. GEORGE TICE)  
San Diego, CA 92152

**MR. STANLEY BROWN** (505) 264-5289  
AFCMD/QA AV 964-5289  
Kirtland AFB, New Mexico 87115

**MR. ALTON PATTERSON** (916) 643-4762  
SM-ALC/MMECP AV 633-4762  
McClellan AFB, CA 95652

**MR. PAUL REIMANN** (801) 777-7231  
Ogden ALC/MMECA AV 458-7231  
Building 1205  
Hill AFB, Ogden, UT 84056

**MR. BOYD WILSON** AV 945-7211  
San Antonio ALC/MMECD  
Kelly AFB, San Antonio, TX 78241

**MR. ROBERT E. ALGER** (205) 532-1257  
Teledyne Brown Engineering  
Research Park  
Huntsville, AL 35087

**DR. ROGER R. BATE** (214) 238-3052  
Texas Instruments Incorporated  
P.O. Box 226015 MS 295  
Dallas, TX 75266

**MR. GENE F. WALTERS** (408) 734-3571  
General Electric Company  
450 Persian Drive  
Sunnyvale, CA 94086

**MS. MARILYN S. FUJII** (213) 831-0611  
Logicon, Inc.  
Strategic and Information Systems Division  
255 West Fifth Street  
P.O. Box 471  
San Pedro, CA 90733

APPENDIX 2 - BIBLIOGRAPHY

## APPENDIX 2 - BIBLIOGRAPHY

The following documents contain policy, requirements, and guidelines that pertained to software acceptance criteria and were consulted by the panel.

1. DOD Directive 5000.29, Management of Computer Resources in Major Defense Systems, 26 April 1976.
2. MIL-STD-480, Configuration Control - Engineering Changes, Deviations and Waivers, 30 October 1968.
3. MIL-STD-490, Specification Practices, revised 18 May 1972.
4. AFR 800-14, Volume I, Management of Computer Resources in Systems, 12 September 1975.
5. AFR 800-14, Volume II, Acquisition and Support Procedures for Computer Resources in Systems, 26 September 1975.
6. MIL-STD-483 (USAF), Configuration Management Practices for Systems, Equipment, Munitions and Computer Software, revised 1 June 1971.
7. MIL-STD-1521A (USAF), Technical Reviews and Audits for Systems, Equipments and Computer Programs, 1 June 1976.
8. MIL-S-52779 (AD), Software Quality Assurance Program Requirements, 5 April 1974.
9. MIL-STD-1679 (NAVY), Weapon System Software Development, 1 December 1978.
10. TADSTAND 9, Software Quality Testing Criteria Standard for Tactical Digital Systems, 18 August 1978.
11. U.S. Army Test and Evaluation Command, Test Operations Procedure, Software Testing, TOP 1-1-056, 15 November 1977.
12. Management Guide to Avionics Software Acquisition, Vol. 2, Software Acquisition Process, AD/A-030-592, Logicon, Inc., June 1975.
13. Software Acquisition Management Guidebook Verification, ESD AD-A048-577 (TR 77-263), August 1977.
14. Airborne Systems Software Acquisition Engineering Handbook:
  - a. Verification, Validation, and Certification, 30323-6009-TU-00, September 1978
  - b. Reviews and Audits, 30323-6006-TU-00, November 1977
  - c. Quality Assurance, 30323-6005-TO-00, November 1977

APPENDIX 3 - LETTER TO PANEL PARTICIPANTS

255 W. Fifth Street, P.O. Box 471  
San Pedro, California 90733  
(213) 831-0611

**LOGICON**

15 March 1979

To: Members of the JLC Panel on Software Acceptance Criteria

Thank you for agreeing to serve on the Software Acceptance Criteria Panel of the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management. It will be a pleasure to serve with such a distinguished group at the Monterey workshop.

By a separate mailing you will receive an agenda for the workshop. I am writing to give you more information on what our specific panel will accomplish. The charter of our panel is to develop a draft document that delineates criteria for the acceptance of embedded computer software, with the goal that this draft would eventually lead to a triservice standard. Rather than attempt to develop a totally new standard in the short time available to us, I propose to use the Navy's TADSTAND 9 (attached) as a baseline document for our work.

I will meet all of you on Monday following the general workshop assembly. Detailed work by our panel should commence on Tuesday morning. At this time, we will discuss panel objectives, review existing practices, and develop a set of detailed problems to be studied. We will then break into subpanels, each of which will have topics to investigate and resolve during the afternoon. On Wednesday morning, each of the subpanels will report its findings to the panel. We will then develop directives for changes to TADSTAND 9. Each of the subpanels will then reconvene with a rewrite assignment for Wednesday afternoon. On Thursday morning, the entire panel will review and approve suggested changes. The results at this time will then be summarized for presentation at the general assembly on Thursday afternoon.

I'd like to ask each of you to give some thought to the problems associated with developing a common standard in this area, and possible solutions to problems. Some areas that I think will require considerable work on our part are:

1. What level and type of unresolved software errors can be tolerated before the software is considered to have failed acceptance?
2. Is the acceptance a one-time event or should there be a formal acceptance of increments throughout the development cycle?
3. How can acceptance criteria be applied for elements of software that are subsequently integrated and reaccepted as a portion of a larger software system or weapon system?

15 March 1979  
Page 2

**LOGICON**

4. How would such a proposed standard interface with other standards that regulate software acquisition management?
5. Should acceptance criteria be limited to "black box" testing or is it appropriate to analyze supporting documentation and design?
6. Can one set of criteria serve as a standard for different types of embedded computer systems or is it necessary to develop a family of criteria?
7. How are vendor/commercially available software packages (e.g., operating systems) that become an integral part of the embedded system treated in accepting operational software?
8. What is the role of the acceptance criteria standard for maintenance of operational software?

In particular, any standards or practices that you know of which relate to this area are solicited. We will have a library at Monterey that will be stocked with the appropriate military standards (e.g., MIL-STD 483, MIL-STD-490, MIL-STD-1521A, MIL-STD-1679, MIL-STD-52779(AD), SECNAVINST 3560.1, various DIDs, guidebooks) for our use.

I look forward to interesting sessions with you all in Monterey. If you have any questions or thoughts, please call me at (213) 831-0611 or the workshop coordinator, Frank Barricelli, at (201) 544-2937.

Sincerely,



R. Dean Hartwick, Manager  
Planning and Development

RDH/pmd

Attachment

**APPENDIX 4 - PANEL HANDOUT**

AGENDA FOR SOFTWARE ACCEPTANCE CRITERIA PANEL

TUESDAY, 3 APRIL 1979

0830 Convene  
0845 Presentation of Service Views  
- J. Gary Nelson / U.S. Army Test and Evaluation  
- Richard Mitchell / Naval Air Development Center  
- LTC Thomas Jarrel / Aeronautical Systems Division  
- Alton Patterson / Sacramento Air Logistics Center  
0945 Review of Objectives and Problems  
1015 Formation of Subpanels  
1030 Coffee Break  
1045 Subpanel Meetings  
1200 Lunch  
1300 Subpanel Meetings  
1630 Subpanel Presentations  
1730 Adjourn

WEDNESDAY, 4 APRIL 1979

0830 Consolidation of Subpanel Findings  
1000 Formation of Rewrite Subpanels  
1030 Coffee Break  
1045 Subpanel Meetings  
1200 Lunch  
1300 Subpanel Meetings  
1630 Subpanel Presentations  
1730 Adjourn

THURSDAY, 5 APRIL 1979

0830      Consolidation of Subpanel Findings  
1015      Coffee Break  
1030      Preparation of Panel Findings  
1130      Lunch

## STANDARDS INTERFACE SUBPANEL

### ASSUMPTIONS -

1. Interfaces with MIL-STDs 1679, 483, 490, 1521, 52779, SECNAVINST 3560.1

### ISSUES -

1. How does proposed standard interface with other existing standards?
2. What are difficulties of generalizing to a triservice applicable standard?
3. What is applicability to maintenance and user organization?

### OUTPUT -

1. Define position of acceptance criteria standard within existing specifications tree.
2. Determine requirements for triservice acceptability.
3. Define applicability to user and maintenance organizations.

## ACCEPTANCE PROCEDURES SUBPANEL

### ASSUMPTIONS -

1. Interfaces with MIL-STDs 1679, 483, 490, 1521, 52779, SECNAVINST 3560.1
2. Interface with specifications tree defined as single document.
3. Well-defined error theory.

### ISSUES -

1. Where in development cycle are acceptance criteria to be applied?
2. Is acceptance testing self-contained or does it include results of intermediate testing?
3. What are test bed requirements?
4. How is software acceptance testing interfaced with system acceptance?
5. Should acceptance testing be limited to black box testing?
6. How are off-the-shelf software components handled?

### OUTPUT -

1. Determine where acceptance criteria are to be applied.
2. Define software acceptance testing vis-a-vis system acceptance.
3. Determine how off-the-shelf software will be accepted.

## SOFTWARE ERROR THEORY SUBPANEL

### ASSUMPTIONS -

1. Standard to be applied by program offices with average or marginal technical expertise.
2. Procedures for application defined elsewhere

### ISSUES -

1. What are meaningful categories of software errors (severity, function, etc.)?
2. What are metrics by which errors are collected?
3. What is the measure of acceptable levels of error?
4. Does error theory vary in different classes of software?

### OUTPUT -

1. Define categories of error.
2. Define levels of error acceptability.
3. Classify software classes to which error theory will be applied.

## **SUBPANEL FINDINGS FORMAT**

- 1. Objectives**
- 2. Issues**
- 3. Analysis**
- 4. Conclusions**
- 5. Recommended TADSTAND 9 changes**

(In all cases, state briefly what recommended change is followed by appropriate cross reference or analysis to justify.)

- 1.1 Changes**
- 1.2 Additions**
- 1.3 Other documents or specification changes required to enact**

APPENDIX 5 - PANEL PRESENTATIONS

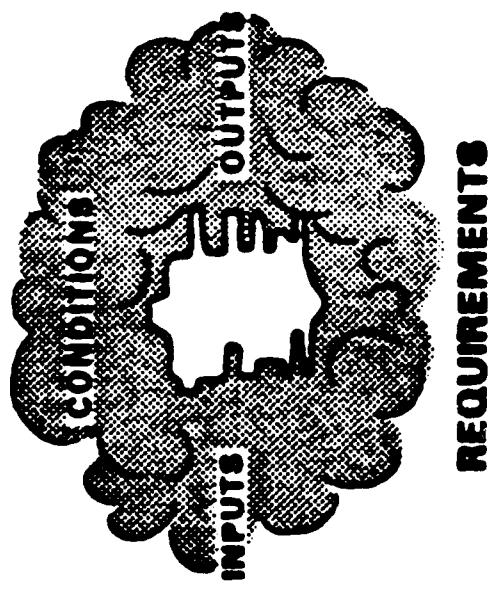
PRESENTATION MADE BY  
J. GARY NELSEN  
U.S. ARMY TEST AND EVALUATION COMMAND  
TO  
SOFTWARE ACCEPTANCE CRITERIA PANEL  
3 APRIL 1979



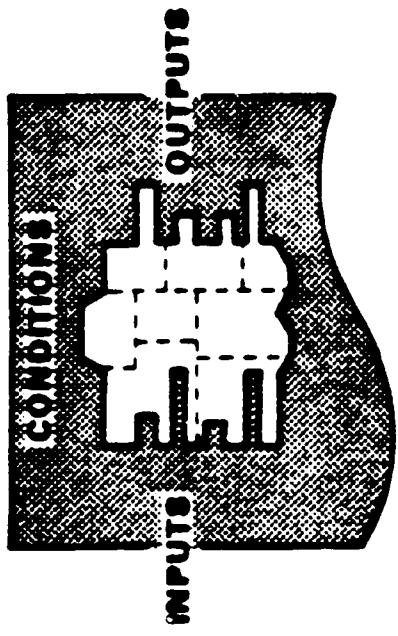
## CONTENTS OF A PERFORMANCE DESCRIPTOR

1. CONCISE REQUIREMENT STATEMENT
2. SOURCE DOCUMENT
3. IMPLEMENTING PROCESS
4. INPUT
5. PROCESSING CONDITION
6. OUTPUT
7. CONSTRAINTS
8. EXECUTION SEQUENCE
9. ERROR RESPONSE
10. PROCESSING VOLUME
11. ACCURACY
12. TIME
13. SIZING
14. CROSS REFERENCE
15. COMMENTS

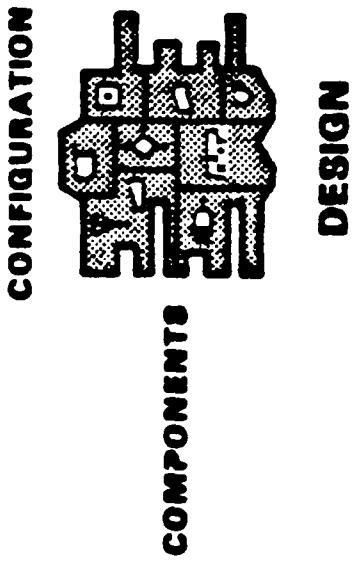
**FORM OF  
REQUIREMENTS, SPECIFICATIONS & DESIGN**



**REQUIREMENTS**

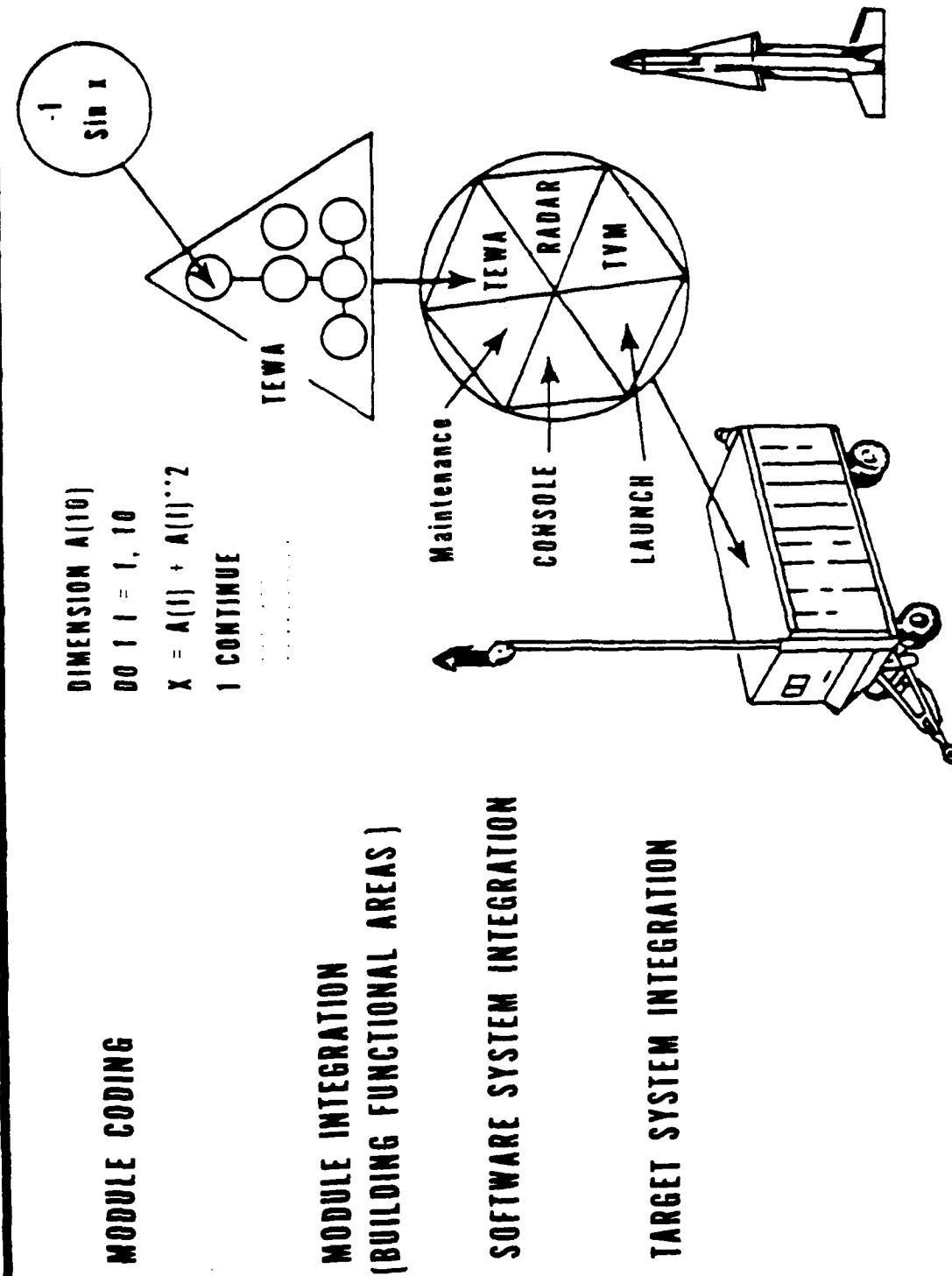


**SPECIFICATIONS**





## PHYSICAL SOFTWARE DEVELOPMENT AND TESTING LEVELS





## ATTRIBUTES OF "GOOD" SOFTWARE SPECIFICATIONS

- NECESSARY \_\_\_\_\_ GOLDPLATING, BOILER PLATE DELETED
- COMPLETE \_\_\_\_\_ ALL INTERFACES, ENVIRONMENTS SPECIFIED
- CONSISTENT\* \_\_\_\_\_ INCOMPATIBILITIES CHECKED & RESOLVED
- TESTABLE\* \_\_\_\_\_ PARAMETER VALUES QUANTIFIED
- TRACEABLE\* \_\_\_\_\_ SPECIFICATION "TREE" CONSTRUCTED
- FLEXIBLE \_\_\_\_\_ MODULARITY SPECIFIED, TRADEOFFS PROVIDED
- FEASIBLE \_\_\_\_\_ CRITICAL ELEMENTS DEMONSTRATED, COSTED

\* ESPECIALLY ADAPTABLE FOR COMPUTER-AIDED VALIDATION

Table B-1. EDWA II Performance Descriptors

EDWA II Activities	DPSR Section	Time	Range & Accuracy	Data Volume	Performance Descriptors		Conditions/ Resources	Interfaces*
					Priority 1	Priority 2		
1.0 New Track Processing	4.1.1	.1 sec max	Integer Integer	1 Target	P-SURV S-DCIP S-DCIP	P-SURV S-DCIP S-DCIP		
1.1 Target Classification								
1.2 Target Identification								
2.0 Identification Change	4.1.2	.5 sec max		1 Target	Priority 2	P-DCIP		
2.1 Hostile Input Processing						S-DCIP		
2.1.1 Non-range resolved target						S-Trk Update		
2.1.2 Range resolved target (set engagement parameters)						S-Trk Update		
2.2 Other input processing (clear engagement parameters)								
3.0 Track Update (Range resolved target only)	4.1.3	1 sec - (until target is engaged or non-hostile)		3 Targets	Priority 3	P-EDWA		
3.1 LNIP Computation				1 Target				
3.1.1 Intercept solution test						S-Trk Update		
3.1.1.1 Engageable rules								
3.1.1.2 Non-engageable rules								
3.1.2 Radar coverage test						S-PRET		
3.1.2.1 Inside coverage rule						S-Trk Update		
3.1.2.2 Outside coverage rule						S-Trk Update		
3.2 PRET Function					3 Targets			
3.2.1 One Engagement (compute TGE)					1 Target			
3.2.2 Other Engagements (compute $\Delta t$ )					3 Targets			

## GENERAL SOFTWARE INSTRUMENTATION AND TEST TOOLS

- STATIC ANALYZERS
- CODE ANALYZERS
- SYMBOLIC EVALUATION SYSTEMS
- SELF METRIC INSTRUMENTATION
- DYNAMIC ASSERTION PROCESSORS
- TEST DATA GENERATORS
- TEST FILE GENERATORS
- EXECUTION VERIFIERS
- OUTPUT COMPARATORS
- TEST HARNESSES
- SOFTWARE MONITORS
- HARDWARE MONITORS
- SYSTEM DRIVERS

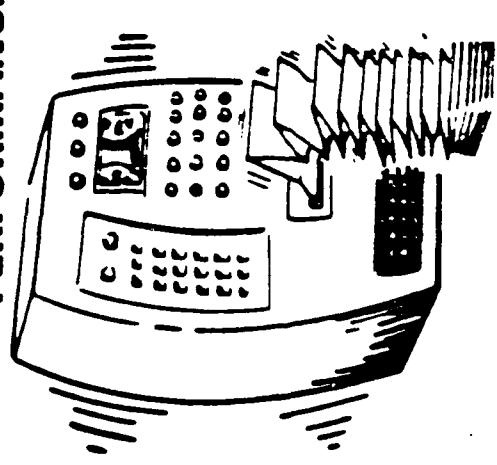


## REQUIREMENTS IDENTIFICATION

- REVIEW ALL DOCUMENTATION
- EXTRACT CONCISE, DIDACTIC STATEMENTS OF WHAT IS TO BE DONE AS THE REQUIREMENT STATEMENT
- ESTABLISH A TRACEABILITY (CAPABILITY, PERFORMANCE DESCRIPTOR) MATRIX

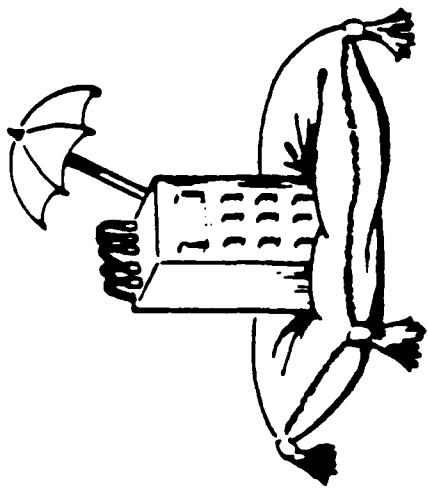
## **COMPUTER SYSTEM REQUIREMENTS**

**INTERFACE**



**PERFORMANCE**

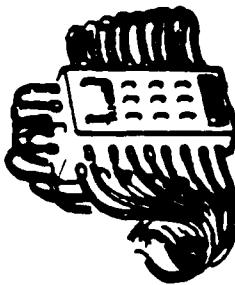
**ENVIRONMENTAL**



**LOGISTICS**

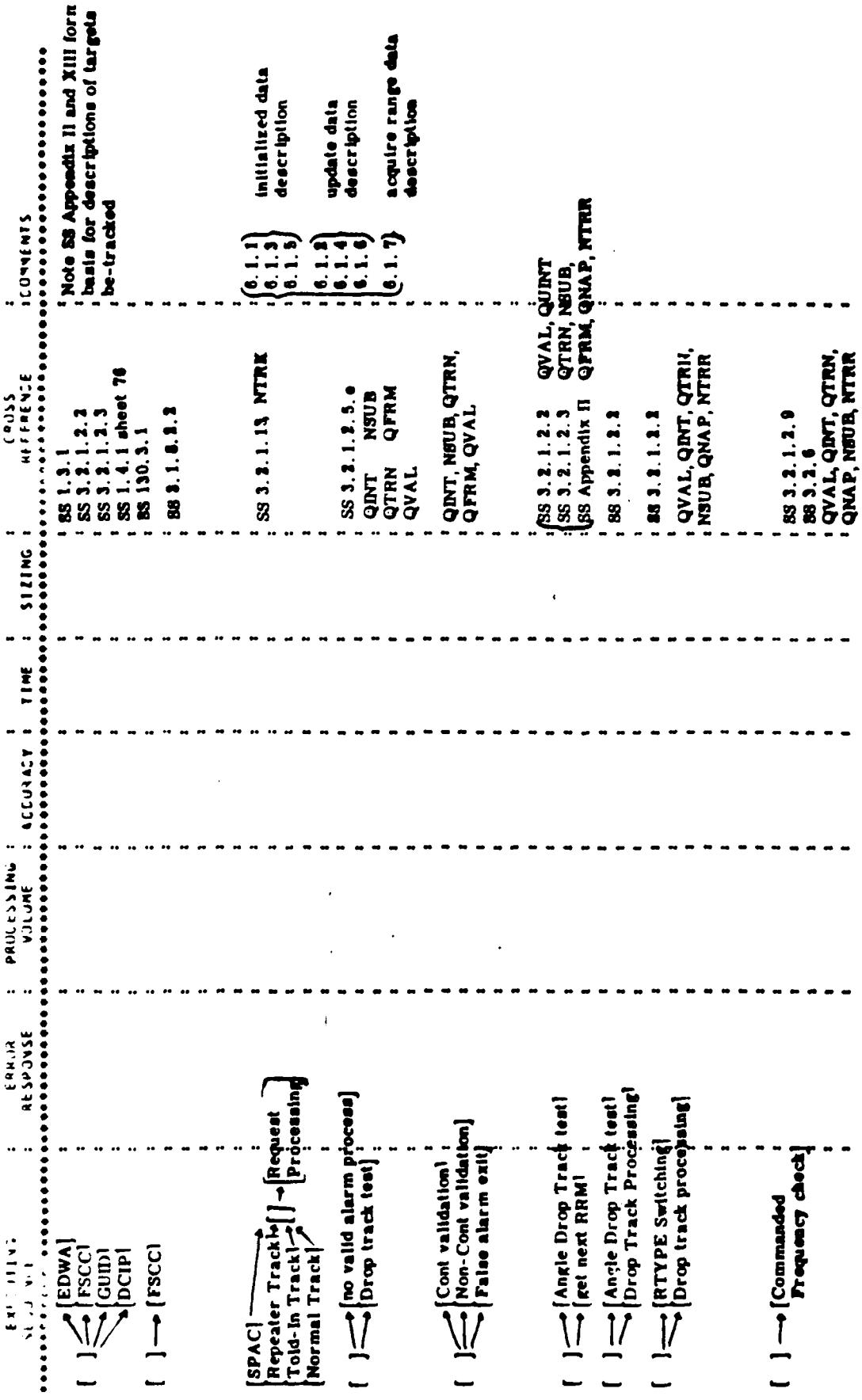
**SPACE & WEIGHT**

**RELIABILITY**



- REQUIREMENTS IDENTIFICATION
- REQUIREMENTS TRACING
- TESTING
- TEST DESIGN
- METHODS AND TECHNIQUES

DPSR	SUB-SECTION	PROCESS	INPUT	PROCESSING	OUTPUT	CONSTRAINTS
1	Initialize and maintain track on targets within a specified volume.	TRAC	• HRM • Target data	• data to initialise or update target info available	• Radar action requests • Target data	
2	Communicate with another sensor.	TRAC	• Communication Msg	• Msg required (see COMMO msg processing for details)	• Communication Msg	Limitation for FCG #1 hr
3	Initialise target data or update target data or, acquire range on angle track only target	Normal track	• RRM • task record • target record	• RRM to be processed	• target data • radar action request	For initialization it is assumed that a default record exists. Updating may be under a SAR rules.
4	Process valid alarms	6.1.1				
5		6.1.2				
6	Process non-valid alarms	6.1.3				
7		6.1.4				
8		6.1.5				
9		6.1.6				
10		6.1.7				
11		6.1.8				
12		6.1.9				
13		6.1.10				
14		6.1.11				
15	Drop target if next predicted position exceeds range limits	6.1.12				
16	Drop track if next predicted position exceeds azimuth/elevation contour bounds unless target is engaged.	6.1.13				
17		6.1.14				
18		6.1.15				
19		6.1.16				
20		6.1.17				
21	Provide RTYPE switching for:	6.1.18				
22	-geometric considerations or system resource constraints -non-beigen RTYPES	6.1.19				
23		6.1.20				
24		6.1.21				
25		6.1.22				



### **TYPES OF SIMULATIONS USED**

- FUNCTIONAL SYSTEM SIMULATION
- COMPUTER SYSTEM SIMULATION
- ENGINEERING SIMULATION

**COMMON FAULTS  
OF  
SYSTEM REQUIREMENTS**

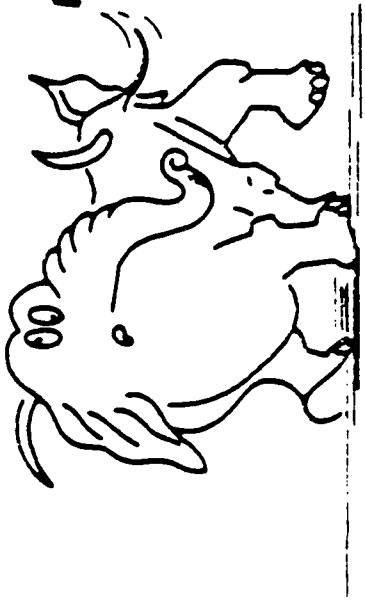
**EXCESSIVE**



**INCOMPLETE**



**INCONSISTENT**



## **MAJOR POINTS I'VE BEEN TRYING TO MAKE**

- THE VERY MOST IMPORTANT JOB IN SOFTWARE DEVELOPMENT IS THE PRODUCTION OF GOOD REQUIREMENTS / SPECIFICATIONS.
- SOFTWARE IS CREATED AND SHOULD BE "ACCEPTED" OVER THE ENTIRE DEVELOPMENT CYCLE.
- UNLESS THE SYSTEM IS TRIVIAL ACCEPTANCE MUST NOT BE BASED SOLELY ON THE "ILLITIES" OF THE IMPLEMENTATION OF HIGHER LEVEL REQUIREMENTS, CARTE BLANCHE
- ACCEPTANCE CRITERIA SHOULD BE BASED ON THE "COMPLETENESS" OF ALL REQUIREMENTS DEMONSTRATION.

## **SAFEGUARD C<sup>3</sup> Capability Matrix**

## **SAFEGUARD PW Capability Matrix**

AN EQUIP. + TEST LAUNCH

SAT PERSONS - Sonnenfeld - 000000

REVISI0N DATA 00100000

PAGE - 31 -

SECTION - 018 -

CAPABILITY

- 19.1.0 TACTICAL LAUNCH AREA TESTS.  
19.1.1 General.  
19.1.1.1 ESTABLISH GROUND COMMUNICATIONS FOR  
ALL MISSILES TO THE OPS.  
19.1.1.2 MAINTAIN THE MISSILE HISTORY TABLE.

- 19.1.1.3 THERE ARE SIX BASIC PROGRAMS: LSPY,  
LSMT, LAC, ALSPY, ALSMU, AND SALS.  
19.1.2 LAUNCH STATION PRACTICAL TEST.  
19.1.2.1 ESTABLISH GROUND WIRE COMMUNICATIONS  
VIA LINCIC TO RELAY CONVEYER.  
ESTABLISH RF COMMUNICATIONS VIA RISA.  
19.1.2.2 AF COMMANDS WILL CHECK OUT THE MISSILE  
GUIDANCE SYSTEM.  
GROUND OPERATORS SEND TO BRING THE  
MISSILE TO THE AF TEST REQUEST STATE.  
A FREQUENCY LOOP SMALL BE CLOSED.  
19.1.2.3  
19.1.2.4 A RANGE LOOP SMALL BE CLOSED.  
ASSURE THAT AMPLITUDE, SINE ALPHA,  
SINE BETA RANGE, FREQUENCY AND WITHIN  
LIMITS.  
19.1.2.5  
19.1.2.6  
19.1.2.7  
19.1.3 ATROPE LAUNCH STATION PERIODIC TEST.  
19.1.3.1 COMPLEMENTARY THE LAUNCH STATION  
PERIODIC TEST.  
19.1.3.2 LOW ALTITUDE SMALL SEND THE RF GATES VIA  
THE INPUT/OUTPUT MANAGERS.  
SWITCH OF COMMAND TRANSMITTER ROUTINES TO  
ENHANCE PROPER TRANSMITTER INTERFACE  
LAUNCH STATION MAINTENANCE TEST.  
19.1.4 LAUNCH STATION WIRE COMMUNICATIONS LINE  
BETWEEN MISSILE SITE DATA PROCESSING  
SYSTEM AND THE MISSILE UNDER TEST.  
19.1.4.1 A FAULT LOCATION IS PERMIT ON A PART  
ISOLATION TEST SET ISPARTANI SMALL BE  
USED IN LIEU OF THE MISSILE.  
DIRECT GATES IN COMMAND COMMUNICATIONS  
AND LAUNCH PREPARATION EQUIPMENT.

- 19.1.4.2  
19.1.4.3  
19.1.4.4  
19.1.4.5  
19.1.5 ATROPE LAUNCH STATION MAINTENANCE  
TESTS.  
19.1.5.1 SAME OBJECTIVE AS LAUNCH STATION MAIN-  
TEINANCE TESTS. THE ONLY DIFFERENCE  
BRING THE FORMAT OF DATA'S TO THE MIS-  
SILE AND REPORTS FROM THE MISSILE.  
19.1.6 LOCAL IN-SATELLITE COMMUTER TEST.

C1-SAI CODE INSPECTION. AC-SAI ALGORITHM CHECKS, STACS-SAI STACKS AND, VISUAL VERIFICATION BY INSPECTOR. GET-CALIBRATION.  
CO-COMPILER, S-AI-PARTITIVE. NO-MOT. I-IMPLD. F-PAILED. F-PAILED. 110-010 FINAL INSTRUM REPORT. 110-010 INSTRUM REPORT. 110-010 GENERATING.



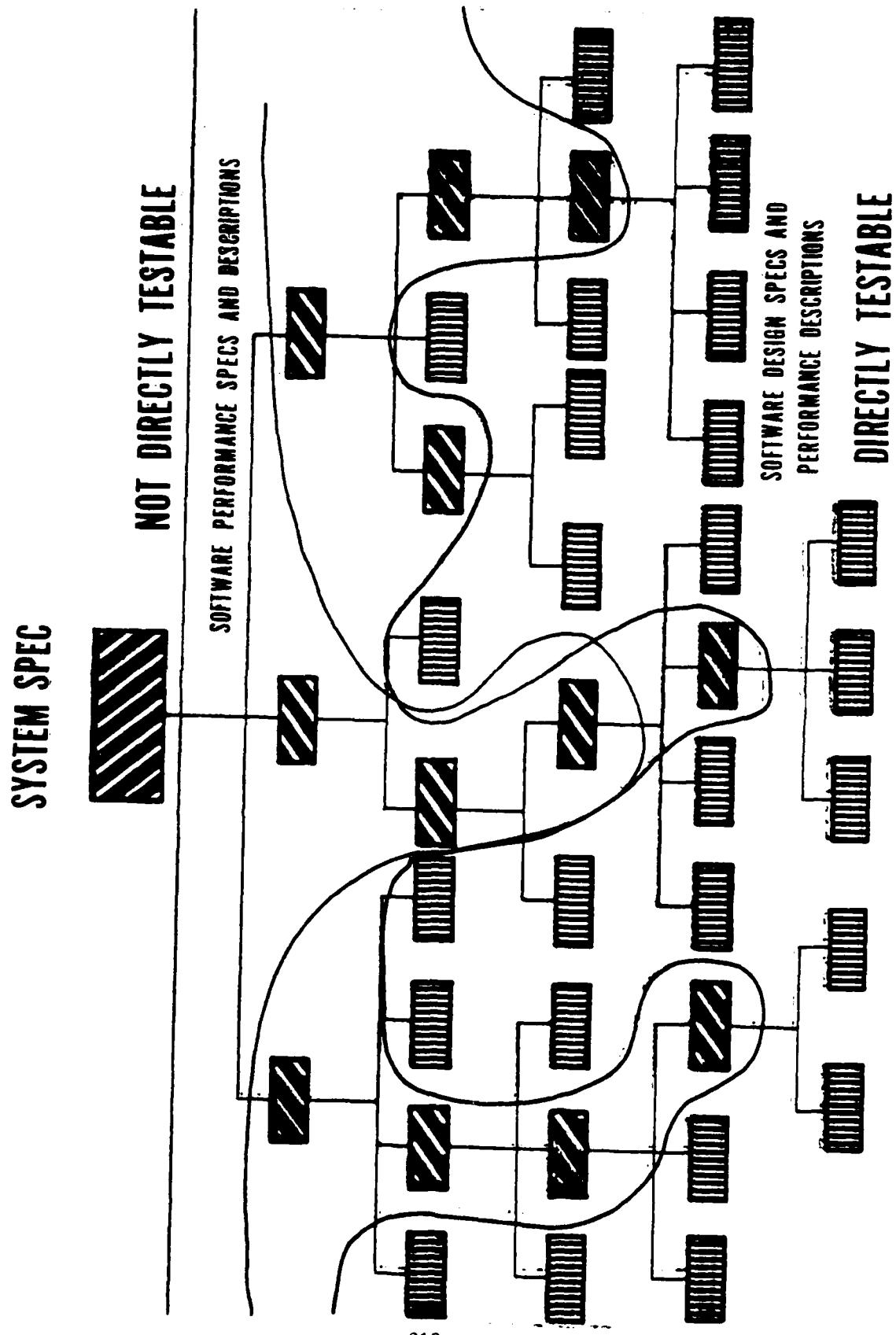
## TOP 1-056

### SOFTWARE TEST OBJECTIVES

1. PARTICIPATE IN SPEC DEVELOPMENT
2. ASSURE ALGORITHMIC CORRECTNESS
3. DETERMINE DOCUMENTATION ADEQUACY
4. MONITOR ALL V&V ACTIONS
5. VALIDATE SIMULATIONS
6. SYSTEMATICALLY DETECT AND ANALYZE SOFTWARE FAILURES
7. DETERMINE RESOURCE CONSTRAINTS AND EXCESSES
8. ASSESS SOFTWARE ASPECTS OF SYSTEM SPEC COMPLIANCE
9. DETERMINE RETEST REQUIREMENTS FOR SOFTWARE "FIXES"
10. MEASURE OPERATING SYSTEM FUNCTIONING
11. RELATE SYSTEM AND COMPUTER TIME LINES
12. INSURE A CONSISTENT SOFTWARE INCIDENT REPORTING SYSTEM
13. SUPPORT SOFTWARE PORTION OF SYSTEM EVALUATION

## **USES OF SIMULATION IN SOFTWARE DEVELOPMENT**

- SIZING STUDIES
- HARDWARE/SOFTWARE TRADE OFFS
- TIMING STUDIES
- SYSTEM PARAMETER INTERRELATIONSHIPS
- DESIGN SENSITIVITIES
- RESOURCE UTILIZATION
- TIME MARKS AND WINDOWS
- COMPUTATIONAL ACCURACIES
- TEST SCENARIO DESIGN
- FAILURE ANALYSIS
- EDUCATIONAL TOOL



TO  
SOFTWARE ACCEPTANCE CRITERIA PANEL  
3 APRIL 1979

FLEET SOFTWARE ENGINEERING/ANALYSIS

MAJOR PROJECTS

LAMPS

S-3A

P-3C

CV-TSC

ASP

FLEET SOFTWARE ENGINEERING/ANALYSIS

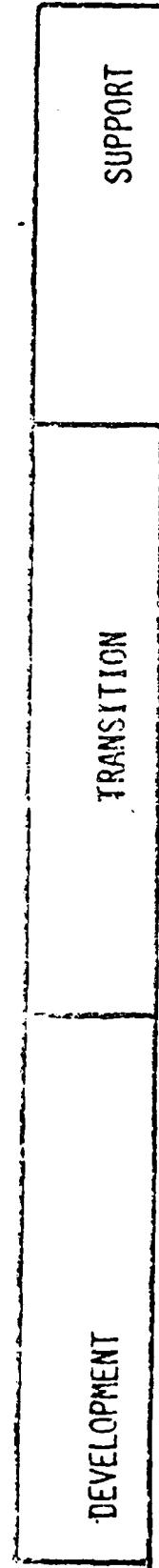
WHAT SOFTWARE IS CURRENTLY INCLUDED

- OPERATIONAL SOFTWARE (APPLICATIONS)
  - TACTICAL
  - SYSTEM TEST
- SUPPORT SOFTWARE
  - SYSTEM GENERATION
  - SIMULATION/STIMULATION
  - HARDWARE INTEGRATION TEST
  - PERFORMANCE ANALYSIS TOOLS

ADDITIONAL SOFTWARE SUPPORT

- AUTOMATIC TEST EQUIPMENT (ATE) SOFTWARE
- TRAINER SOFTWARE (NST)
- EMBEDDED COMPUTER SOFTWARE
- FIRMWARE/MICROCODE

FLEET SOFTWARE ENGINEERING/ANALYSIS



SUPPORT TO DEVELOPMENT

- TECH ADVISOR (I.E. REQ) • CM
- SUPPORT DESIGN REVIEWS • QA
- REVIEW MAIN. PHILOSOPHY • PTR ANALYSIS
- CM PLANS • PTR TRACKING • CM
- DOCUMENTATION REVIEW • TRAINING SUPPORT PERSONNEL • QA
- FACILITIES REQUIREMENTS • DOC REVIEW VALIDATION • DOCUMENTATION
- FACILITIES DEVELOPMENT • FACILITIES OPERATION

FLEET SOFTWARE ENGINEERING/ANALYSIS

RESOURCE UTILIZATION

- ESTABLISH/MAINTAIN CONFIGURATION BASELINE
- PLACE IN-HOUSE EMPHASIS ON FRONT DESIGN, IN-PROCESS PERFORMANCE MEASUREMENTS AND PRODUCT CERTIFICATION
- DEVELOP FUNCTIONAL GROUPS WITHIN THE DIVISION
  - CM : NAVIGATION
  - QA : COMMUNICATION
  - FACILITIES : TACTICS

## FLEET SOFTWARE ENGINEERING/ANALYSIS

### USE OF LAB FACILITIES

#### PROGRAM DEVELOPMENT

- CODE GENERATION
- CODE DEVELOPMENT
- DEBUG
- INTEGRATION
- T&V

#### PROGRAM SUPPORT

- CM
- V&V
- TRAINING
- ENGINEERING ANALYSIS

## FLEET SOFTWARE ENGINEERING/ANALYSIS

PRIME FACTORS CONTRIBUTING TO LIFE CYCLE COST:

REQUIREMENTS DEFINITION/CHANGES

SYSTEM FUNCTIONAL DEFINITION/ALLOCATION

CONCURRENT HARDWARE DEVELOPMENT

PROGRAM DESIGN/CONSTRAINTS

SUPPORT SOFTWARE/DOCUMENTATION

TECHNICAL STAFF TRAINING/STABILITY

COST/SCHEDULE CONTROL

## FLEET SOFTWARE ENGINEERING/ANALYSIS

STEPS TAKEN TO REDUCE LIFE CYCLE COST:

- PROVIDE SOFTWARE GFE
- PROVIDE FACILITIES/FACILITIES SERVICES GFE
- DEVELOP AND CONTROL REQUIREMENTS
- DEVELOP AND/OR PROVIDE CRITICAL ALGORITHMS
- CONDUCT DESIGN REVIEWS
- PLAN INCREMENTAL PROGRAM BUILD/C
- IMPLEMENT SOFTWARE G4
- PROVIDE LIBRARY AND DOCUMENTATION SUPPORT
- PROVIDE CLOSE CONTRACTOR/DEVELOPER/USER INTERFACE
- USE INDEPENDENT TEST

MINUTES OF PRESENTATION MADE TO  
SOFTWARE ACCEPTANCE CRITERIA PANEL

3 APRIL 1979

Dick Mitchell, of the Naval Air Development Center, discussed the Navy's view. His organization is responsible for development all the way through life cycle support. It handles major weapon systems that range from 1/4 to 1-1/2 million-word programs, plus support software—mostly ASW, but including all avionics, and including trainers. This software is provided GFE (often to contractors). NADC builds tapes from the Center, and also requires Contractors to use NADC facilities in development software. Contractors are largely used but NADC has set baselines and system requirements since P3 system was introduced and subsequent major updates have been made. The software baseline has been maintained. NADC uses incremental builds, etc.

NADC is also able to build up in-house expertise in technical areas such as navigation. NADC worries less about whether the code is correct, than about whether it properly fulfills the functions. NADC provides contract GFE algorithms to the contractor. NADC requires that contractors relocate within 10 minutes of NADC, to assure continuing contact; sometimes contractors have their own development stands, but if so, an equivalent exists at NADC. Each project will have its own integration stands, but sometimes there is a crunch in getting computer support, etc.

NADC emphasizes cost and control of support software, and reduces cost in providing much of the support software and algorithms as GFE. The contractor's use of NADC facilities is itself recorded; he cannot even hide the work on a particular module. NADC conducts design reviews and configuration management, and designs its own simulators to see that contractors correctly implement NADC-supplied algorithms, etc. Naval T&E (Test and Evaluation) group does V&V on NADC - developed software, and Naval T&E is involved, as with other users, from the start. NADC feels that acceptance criteria are relatively easy if the work up front has been done. NWC (China Lake) is the agency responsible for F-18 software. It does not have "SITS (software integration and test site) bench" on the F-18, as there was on the F-14. NADC claims that if the prime contractor does all of the system, some acceptance criteria might not apply as they would if NADC did the system. General audience consensus was that NADC has an ideal setup in its heavy involvement in development. One stated danger is that the Navy cannot sue the Navy if its Specs or code have errors.

Alton Patterson gave an Air Force viewpoint on acceptance on software. The criteria of acceptance probably could be agreed upon, but the method of evaluating these criteria's fulfillment, and enforcement, may differ. Al presented an AFLC perspective, from the standpoint of a maintenance problem (after deployment of the system). Software suitability (maintainability and usability) includes qualification criteria, specs, and performance of requirements, but AFLC has "microcosms" of updating. About 100 people support the F-111 OFP (fire control). They use simulation and flight tests to that end. A "Strawman's approach" looks at error sources, and software acceptance considers those error sources. The applications of end-item software test give a degree of confidence in that measure of software correctness. But to handle the 5,000 configured support software items, no such extensive tests are possible to the degree of end-item (F-111 OFP) checks.

Software acceptance criteria are not all satisfied at once. Problems exist with incorrect specs, or with coding not following specs. Function breakouts are in increasing levels of detail (such as time-dependent operations versus those that are not). The F-111 correctness efforts are focused on externally-caused errors of computer hardware, interactive hardware, incorrect human inputs, interface incompatibilities, and incorrect procedures, versus design and coding errors (syntax, semantics, logic, and algorithmic errors). Mr. Pettersen said the Systems Command must put out a good product, but AFLC must worry more about suitability, usability, and maintainability.

Acceptance requirements at Sacramento ALC:

1. Short term:
  - a. Does the System meet spec requirements? (Mode-by-mode functional basis; overall system basis; system integrations; state (Lab) environment.)
  - b. Does the system meet user requirements? (Simple to use; providing effective indications of normal and abnormal (self-test) indications; backup modes adequate.)
2. Long term: Does the system meet maintainability results - sufficient documentation, and modularity, common modules, optimum memory use; development modification/qualification tools available and usable for maintenance; training available; programming; are any portions proprietary? What language used (HOL, easy assembly, etc.)?

All F-111 requirements which are 5 years past deployment stage), result from user requests, plans for new stores, etc. Software is contingently accepted, depending upon final system performance.

Maintainability criteria discussed:

Removal/correction of latest errors; adding new features/capabilities; deletion of unused/undesirable features; software modules to follow hardware changes. This requires software to be: modular, descriptive, consistent, simple, expandable, testable. Mr. Peterson agreed that continual tradeoffs between these items are required. F-111 came after B-5 and C-5 standards were imposed and functions adequately without these particular documents, and allows tailoring to specific systems. Mr. Patterson also said that SMALC does have someone in development of modules, and has IV&V for the Air Force for modules on the F-111 A/E. He said they also can get documentation according to their needs; and said that the SOW to be laid on GD will reflect SMALC's views of software acceptance criteria. If the SPO does not act on SMALC's complaints, the SPO will have to come to terms later with SMALC before turning it over. Al said that examination of documentation, etc., is in effect also part of the testing, which establishes acceptance criteria. SMALC direction comes from Air Staff.

APPENDIX 6 - VISUAL AIDS OF PANEL PRESENTATION TO WORKSHOP

**PRESENTATION OF PANEL OBJECTIVES**

**MADE BY**

**R. DEAN HARTWICK**

**TO**

**SOFTWARE WORKSHOP**

**2 APRIL 1979**

**JLC PANEL ON SOFTWARE ACQUISITION CRITERIA**

**GOAL:**

DEVELOP A DOCUMENT THAT DELINEATES STANDARD CRITERIA  
FOR ACCEPTANCE OF SOFTWARE

**METHOD:**

- USE NAVY'S TADSTAND<sup>9</sup> AS BASELINE
- FORMULATE LIST OF:
  - OBJECTIVES
  - PROBLEMS WITH EXISTING PRACTICES
- ANALYZE EXTENT TO WHICH TADSTAND<sup>9</sup> SATISFIES OBJECTIVES  
AND PROBLEMS
- MODIFY TADSTAND<sup>9</sup> TO ACCOMODATE OBJECTIVES AND SATISFY  
PROBLEMS

## **SOFTWARE ACCEPTANCE TESTING - ISSUES**

### **STANDARDS INTERFACE**

- HOW DOES PROPOSED STANDARD INTERFACE WITH OTHER EXISTING STANDARDS?
- WHAT ARE DIFFICULTIES OF GENERALIZING TO A TRI-SERVICE APPLICABLE STANDARD?
- WHAT IS APPLICABILITY TO MAINTENANCE AND USER ORGANIZATIONS?
- ARE EXISTING ACQUISITION STANDARDS ADEQUATE

## SOFTWARE ACCEPTANCE CRITERIA - ISSUES

### ACCEPTANCE PROCEDURES

- WHERE IN DEVELOPMENT CYCLE ARE ACCEPTANCE CRITERIA TO BE APPLIED?
- IS ACCEPTANCE TESTING SELF CONTAINED OR DOES IT INCLUDE RESULTS OF INTERMEDIATE TESTING?
- WHAT ARE TEST BED REQUIREMENTS?
- HOW IS SOFTWARE ACCEPTANCE TESTING INTERFACED WITH SYSTEM ACCEPTANCE?
- SHOULD ACCEPTANCE TESTING BE LIMITED TO BLACK BOX TESTING?
- HOW ARE OFF-THE-SHELF SOFTWARE COMPONENTS HANDLED?

SOFTWARE ERROR THEORY

- WHAT ARE MEANINGFUL CATEGORIES OF SOFTWARE ERRORS (SEVERITY, FUNCTION, ETC.?)?
- WHAT ARE METRICS BY WHICH ERRORS ARE COLLECTED?
- WHAT IS THE MEASURE OF ACCEPTABLE LEVELS OF ERRORS?
- DOES ERROR THEORY VARY BY DIFFERENT CLASSES OF SOFTWARE?

**PRESENTATION OF PANEL RESULTS**

**MADE BY**

**R. DEAN HARTWICK**

**SOFTWARE WORKSHOP**

**3 APRIL 1979**

SOFTWARE ACCEPTANCE CRITERIA PANEL - STATEMENT OF PROBLEM

PROVIDE FOR A WAY TO MEASURE THAT SOFTWARE AS FIELDDED  
WILL WORK ACCEPTABLY.

SOFTWARE ACCEPTANCE CRITERIA PANEL

CONSENSUS

1. DESIRABILITY OF EARLY REQUIREMENTS/SPEC DEVELOPMENT
2. INDEPENDENCE OF T&E AGENCY
3. ACCEPTANCE DONE IN PARALLEL THROUGHOUT DEVELOPMENT CYCLE
4. SOFTWARE INCLUDES CODE PLUS DOCUMENTATION
5. LEVELS OF SOFTWARE ACCEPTANCE AND CRITERIA ADDRESSES ALL LEVELS

SOFTWARE ACCEPTANCE CRITERIA PANEL - SUBPANEL CHARTERS

ERROR THEORY

- o DEFINE ERROR CATEGORIES
- o DEFINE LEVELS OF ACCEPTABILITY
- o CLASSIFY TYPES OF S/W

PROCEDURES

DEFINE ACCEPTANCE APPROACH AS APPLIED TO DEVELOPMENT  
MILESTONES

STANDARDS INTERFACE

IDENTIFY WHERE S/W ACCEPTANCE CRITERIA ALREADY EXISTS IN STD'S,  
STD'S, REGS, & SPECS AND RECOMMEND MANDATING DIRECTIVE

PRESENTATION OF PANEL RESULTS  
MADE BY  
R. DEAN HARTWICK  
TO  
SOFTWARE WORKSHOP  
4 APRIL 1979

SUB-PANEL ON STDS, SPECS, & REGS

CHARTER

1. REVIEW EXISTING STDS, SPECS, & REGS TO IDENTIFY SOFTWARE ACCEPTANCE CRITERIA PRESENTLY EXISTING
2. IDENTIFY A SUBSET OF THESE WHICH WOULD BE SUFFICIENT (WITH MODS IF NEEDED) FOR USE BY PMS, SPOS, ETC.
3. RECOMMEND JOINT OR DOD MANDATING THE USE OF SUCH CRITERIA.

WORK REMAINING

REVIEW THE STDS, SPECS, ETC. IN SOME DETAIL.

ACCOMPLISHMENTS

1. DRAFTED A TYPICAL, 4-PHASE DEVELOPMENT CYCLE FOR SOFTWARE.
2. DEFINED 4 ACTIVITIES IN EACH PHASE.
3. IDENTIFIED SOFTWARE PRODUCTS AND REVIEWS ASSOCIATED WITH EACH PHASE.
4. FOUND REFERENCES IN EXISTING STDS, SPECS, AND REGS WHERE APPROPRIATE ACCEPTANCE CRITERIA AND REVIEW ACTIVITIES ARE STATED.

WORK NEEDED

1. REFINEMENT OF ABOVE.
2. DRAFT JOINT POLICY WHICH WOULD REQUIRE PMS, SPOS, ET AL TO USE THE ACCEPTANCE STANDARDS WHICH PRESENTLY EXIST.

**ACCEPTANCE PROCEDURE SUBPANEL**

- o MAP S/W ACCEPTANCE CRITERIA OVER SW DEV. MILESTONES

**MILESTONES**

**CRITERIA**

FUNCTIONAL BASELINE DEF OF SW FUNCTIONS  
"A" SPEC

**ALLOCATED BASELINE**

- o CORRECT/COMPLETE ALLOCATION
  - "B" SPEC
  - o FUNCTION/INTERFACE DEF
- SW REQTS
- SW PDR
- SW CDR
- SW FCA/PCA
- SYS FCA/FQR
- DT&E
- OT&E

MILESTONES ADDRESSED

- o FUNCTIONAL BASELINE
- o ALLOCATED BASELINE
- o PDR
- o CDR
- o SOFTWARE FCA
- o SOFTWARE PCA
- o SYSTEM FCA (FQR)
- o DEVELOPMENT TEST
- o OPERATIONAL TEST

ACCEPTANCE CRITERIA

- o DATA FORMAT/STD'S
- o TECHNICAL CONTENT TO SUPPORT SUBSEQUENT DEV STAGES
- o TRACEABILITY BACKWARDS
- o MANAGEMENT
- o LIFE CYCLE SUPPORT ISSUES

## SOFTWARE ERROR THEORY

- o ESTABLISH ACCEPTANCE CRITERIA INDEPENDENT OF METHODOLOGY
  - o APPLY ACCEPTANCE CRITERIA ON EACH MILESTONE
  - o ERROR CATEGORIES FOR SPECS
    - NECESSARY FNS
    - COMPLETE INFO NECESSARY TO PERFORM FUNCTIONAL SIMUL.
    - CONSISTENT
    - TESTABLE
    - TRACEABLE
    - ALGORITHM
  - o LEVELS TO BE ESTABLISHED BY PROGRAM OFFICE FOR SPECIFIC APPLICATION
  - o FIRST 4 CAT. ALSO APPLY TO USER'S MANUAL
- (HAVEN'T GOT TO CODE ACCEPTANCE AND ASSOCIATED ERRORS YET)

SOFTWARE ERROR THEORY

- o ERROR CATEGORIES HAVE BEEN ESTABLISHED FOR
  - SPECIFICATIONS
  - CODE
  - DATA BASE
- o RECOMMEND USING CLASS I ECP JUSTIFICATION CODES FOR IDENTIFYING

OBSERVATIONS:

- o CURRENT THEORY IS STATISTICAL APPROACH BASED UPON A LARGE NUMBER OF LARGE SCALE PROJECTS
- o QUESTIONABLE AS TO HOW IT CAN BE APPLIED IN NEXT 5 YEARS
- o HOWEVER, - ELEMENTS CAN HELP IN DETERMINING ACCEPTABILITY

ELEMENTS OF ERROR THEORY THAT CAN BE APPLIED

- o ERROR CATEGORIES ESTABLISHED FOR EVALUATING CAUSE AND SEVERITY OF ERRORS
- o ERROR DETECTION TRENDS TO DETERMINE WHEN AN ACCEPTABLE LEVEL HAS BEEN ACHIEVED TO MOVE FROM ONE STAGE OF TESTING TO THE NEXT

THURSDAY

1. CONSOLIDATION OF SUBPANEL FINDINGS
  - o ERROR CATEGORIES
  - o ERROR IMPACT
  - o MILESTONE ACCEPTANCE CRITERIA
  - o MIL-STD QA SECTIONS
2. DEVELOP NEAR TERM RECOMMENDATIONS
  - o POLICY STATEMENT
  - o GUIDE TO EXISTING STDS
3. DEVELOP LONG TERM REC
  - o JOINT GUIDEBOOKS
  - o ERROR DATA COLLECTION

PRES EN TATION OF PANEL RESULTS

MADE BY

R. DEAN HARTWICK

TO

SOFTWARE WORKSHOP

5 APRIL 1979

## SOFTWARE ACCEPTANCE CRITERIA

### SUMMARY

1. SET GOALS FOR PANEL
2. DERIVED BASIC CONSENSUS ON PRESENT GOOD PRACTICE
3. ANALYZED AND SET CRITERIA FOR:
  - ERROR CATEGORIES
  - PROCEDURAL APPLICATIONS
4. LOCATED APPROPRIATE SECTIONS OF CURRENT STANDARDS THAT PRESENTLY ENABLE S/W ACCEPTANCE CRITERIA (AND SOME CHANGES)

**SOFTWARE ACCEPTANCE CRITERIA - SUMMARY**

TO BE DONE

1. WRITE DRAFT REPORT
2. WRITE DRAFT TRI-SERVICE POLICY
3. WRITE DRAFT INTERIM GUIDEBOOK

SOFTWARE ACCEPTANCE CRITERIA - RECOMMENDATIONS

1. TRI-SERVICE POLICY TOWARD S/W ACCEPTANCE SHOULD BE TO FOSTER  
ACCEPTANCE TESTING CONTINUOUSLY AND APPLY ACCEPTANCE CRITERIA  
INCREMENTALLY THROUGHOUT SOFTWARE LIFE CYCLE.

**SOFTWARE ACCEPTANCE CRITERIA - RECOMMENDATIONS**

**SHORT TERM**

2. JLC ADOPT POLICY THAT MANDATES USE OF S/W ACCEPTANCE CRITERIA
3. JLC PREPARE INTERIM GUIDEBOOK PANEL FINDINGS THAT PROVIDES GUIDANCE AT SOFTWARE MANAGEMENT LEVEL TO IMPLEMENT POLICY  
MANDATE

SOFTWARE ACCEPTANCE CRITERIA - RECOMMENDATIONS

4. JLC SPONSOR SUBSET OF S/W ACC. CRIT. PANEL TO MEET IN  
LOS ANGELES WEEK OF MAY 21 - MAY 25 TO FINALIZE:

DRAFT REPORT

DRAFT POLICY

DRAFT GUIDEBOOK

**SOFTWARE ACCEPTANCE CRITERIA - RECOMMENDATIONS**

**LONG TERM**

5. JLC SHOULD DEVELOP A UNIFIED SET OF ACQUISITION STANDARDS  
TO BE USED BY ALL SERVICES.
6. THE UNIFIED STANDARDS SHOULD EXPLICITLY INCLUDE S/W  
ACCEPTANCE CRITERIA BASED ON PANEL FINDS.

7. A FINAL TRI-SERVICE GUIDEBOOK BE DEVELOPED BASED ON INTERIM VERSION THAT GIVES PRACTICAL WORKING LEVEL KNOWLEDGE OF APPLYING S/W ACCEPTANCE CRITERIA (UNIFIED OR PRESENT STANDARDS).

**SOFTWARE ACCEPTANCE CRITERIA**

GOAL : DEVELOP RECOMMENDATIONS FOR JLC JPCG-CRM FOR FIRST STEP TOWARD  
TRI-SERVICE STANDARD FOR SOFTWARE ACCEPTANCE CRITERIA

**CONCLUSION:**

1. CRITERIA NEEDED
2. SHORT TERM FIX RECOMMENDED
3. PROCEDURE TO PREPARE SHORT TERM FIX RECOMMENDED
4. LONG TERM FIX RECOMMENDED

APPENDIX 7 - DRAFT OF PROPOSED TRISERVICE POLICY FOR SOFTWARE  
ACCEPTANCE CRITERIA

DRAFT LETTER OF INSTRUCTION  
FOR IMPLEMENTING  
SOFTWARE ACCEPTANCE CRITERIA  
IN  
DEFENSE EMBEDDED COMPUTER SYSTEMS

Subject: Applying software acceptance criteria for embedded computer systems

Enclosure(1): Interim software acceptance criteria guidebook

1. Purpose

The purpose of this letter is to provide instruction and guidance on the use of criteria for accepting computer software for defense systems acquisition.

2. Background

- a. Currently, no definitive criteria for the acceptance of embedded computer software are provided through the existing military standards. This frequently results in a problem for the acquisition program manager in that he has no way to assure that computer software can be accepted with the knowledge that it will perform operationally at an acceptable level. Successfully supplying such criteria not only should satisfy the primary objective of delivering operational software that works, but also should:
  - (1) Allow the extent of acceptable developmental progress to be quantitatively measured.
  - (2) Improve visibility into the developmental status of software throughout the developmental cycle.
  - (3) Provide a better basis for software acquisition managers and software developers to agree on job completion criteria.
  - (4) Provide a basis to better express the quality of delivered software.
- b. Software acceptance criteria are required to measure embedded computer software performance and ensure that the software will perform acceptably in the field. The existing military standards, directives, and regulations provide the necessary authority for a software development manager to apply software acceptance criteria.

- c. Software acceptance criteria should be applied throughout the software acquisition cycle. It is most important that successful software acquisition efforts ensure that the original software requirements are acceptable in order to avoid costly schedule slips at later times caused by correcting deficiencies due to poor statements of requirements. Likewise, a more cost-effective acquisition effort results from assurance that intermediate software products are correctly developed.
- d. Software, for the purposes of applying software criteria to embedded computer systems is assumed to be:
  - (1) All code, including microcode (firmware)
  - (2) All documentation that specifies this code, describes how to use it operationally, and is required to test it
  - (3) Any data resident with the code including constants, parameters, and initial data sets delivered with it.

3. Requirement

Effective the date of this letter, software acceptance criteria shall be applied to all embedded computer software that shall hereafter be acquired. The software acceptance criteria shall be imposed consistent with the software acquisition management standards being used to acquire the software. References to sections of specific applicable software acquisition standards are contained in Table 1 of Panel D's report.

The enclosure is a guidebook that can be used to assist in applying software acceptance criteria under this requirement.

APPENDIX 8 - OUTLINE AND DRAFT MATERIAL FOR PROPOSED  
TRISERVICE INTERIM GUIDELINE ON APPLICATION  
OF SOFTWARE ACCEPTANCE CRITERIA

## INTRODUCTION

The purpose of this guide is to provide the manager of embedded computer software acquisition and maintenance efforts with the means of applying software acceptance criteria. This guide is written for the manager of such efforts who is assumed to have a working knowledge of the acquisition standards and regulations to be used within his area and to possess a technical background. It is not assumed that the manager has an in-depth background in software engineering. Rather, the criteria that are presented are an expression of current software engineering methodology and are delineated here for two purposes: to provide a common checklist for many developments, and to aid less experienced managers in understanding and applying them.

These criteria are defined for use in any effort that is being conducted under existing acquisition standards and regulations. It will accommodate all such standards and regulations currently being used to procure embedded computer systems. (It is likely that it could be used for general ADP situations, but these systems were not considered in developing this guide.) These criteria can be applied to contractual software developments, governmental agency developments, and either contractual or governmental software maintenance.

At the present time, this guide has addressed only eight generic software products. These are:

1. System level specifications
2. Development (or performance requirements) specification
3. Product (or product design) specification
4. Computer program contract item (CPCI) (code)
5. CPCI test plan
6. CPCI test procedures
7. CPCI test report
8. Handbooks and manuals.

Although additional material does pertain to the acquisition process (for example, quality assurance and configuration management plans), these pertain to the way the software effort is managed as opposed to being descriptive of the software product itself.

The following sections contain the software criteria and proposed implementation instructions. The sections of existing standards that can be used to apply acceptance criteria are referenced in "Enabling Standards." Acceptance criteria that can be applied to the eight defined software products are contained in "Software Acceptance Criteria." In this section also is a guide to the time and emphasis when the criteria are applied. In "Action Procedures," alternate action procedures to be followed upon failure of criteria are presented. A glossary of terms is presented in Appendix 9.

## ENABLING STANDARDS

This guide is intended to be applicable for all acquisition standards currently being used by the three services. In order to use this guide for all such standards, it is necessary to define an ideal software acquisition cycle that can be used as a standard of comparison to the other standards. Accordingly, an ideal four-phase embedded computer acquisition cycle is defined. This ideal cycle consists of defining software into one of four development phases:

1. Systems requirements development
2. Preparation of specifications and design
3. Development of the program
4. Evaluation.

With each of these phases, four classes of activity are defined that suggest an event that is to be approved or disapproved. This ideal cycle is shown schematically in Table 8-1. Within each of the activities, the software products typical of that set of activities for each phase of development are defined. The four sets of activities include the five identified prime software products (under documentation) and the three identified test items (under test and evaluation). The other two activities (reviews and audits and configuration management) relate management and control items to the software products. In addition, other software products are defined that are not currently addressed by this guide (for instance, interface design specification).

Each of the activity items within the ideal cycle is cross-referenced in Table 8-2 to the sections of applicable standards where the acceptance criteria are enabled for that particular item. For example, the item for reviews and audits during the evaluation phase is designated a formal qualification review under the idealized cycle definition. The enabling instructions for this review or its equivalent are contained in MIL-STD-1521, paragraph 5.11.1.2 of MIL-STD-1679, and so on. The program in its deliverable form is defined under documentation during the development phase. Appropriate cross-references then are made to Sections 60.5.5 and 60.5.4.2 of MIL-STD-483 and to Section 5.0 of MIL-STD-5277G. This table can be used to gain the authority to impose the appropriate acceptance criterion.

## SOFTWARE ACCEPTANCE CRITERIA

Software acceptance criteria are defined in this section for the eight software products. Each of these products has a set of acceptance criteria defined for it (Tables 8-3 through 8-10). These criteria are to be applied across the seven reviews and audits defined in Table 8-1. The need to approve their satisfaction varies depending upon the product, as shown by the "approval level" given for each of the software products as a function of emphasis that should be placed on each criterion at each review/audit. The approval level ranges in importance from "0" (least prominent at that milestone) to "3" (most prominent at that milestone). The scale is applied according to:

Table 8-1. Ideal Software Acquisition Cycle

Functional Baseline	Allocated Baseline	Product Baseline	Operational Baseline
Systems Requirements	Specification and Design	Development	Support
Development	Development	Development	Maintenance
Yes <u>Reviews and Audits</u> (approve/disapprove)	No ◆ Yes <u>Reviews and Audits</u> (approve/disapprove) System Requirements (SRR)	No ◆ Yes <u>Reviews and Audits</u> (approve/disapprove) Systems Design (SDR) Preliminary Design (PDR)	No ◆ Yes <u>Reviews and Audits</u> (approve/disapprove) Formal Qualification (FQR)
<u>Documentation</u> (approve/disapprove)	<u>Documentation</u> (approve/disapprove)	<u>Documentation</u> (approve/disapprove)	<u>Documentation</u> (approve/disapprove)
System Specification	Program Specification	Program Design Documents	Operator's Manuals
Software Development Plan	Program Design Specification	Program Package	User's Manuals
Software Quality Assurance Plan	Interface Design Specification	Data Base Document	
Project Management Plan			
Integrated Logistics Plan			
Support Software Plan			
<u>Configuration Management</u> (approve/disapprove)	<u>Configuration Management</u> (approve/disapprove)	<u>Configuration Management</u> (approve/disapprove)	<u>Configuration Management</u> (approve/disapprove)
CM/CSA Plan	CM/CSA	CM/CSA	CM/CSA
	Software Errors	Software Errors	Software Errors
	Software Patches	Software Patches	Software Patches
<u>Test and Evaluation</u> (approve/disapprove)	<u>Test and Evaluation</u> (approve/disapprove)	<u>Test and Evaluation</u> (approve/disapprove)	<u>Test and Evaluation</u> (approve/disapprove)
Test and Evaluation Plan	Subprogram Tests	Performance Tests	Software Integration Test
	Function Tests		System Integration Test
			Acceptance Tests
			Technical/Operational Evaluation

Table 8-2. Correlation of Standards and Ideal Cycle (1 of 4)

A. System Requirements Phase Sections

<u>Rev./Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<u>Revised</u>	<u>52779</u>	<u>480</u>
Sys Rqts (SCR)		App. A					
Sys Design (SDR)		App. B					
Prel Design (PDR)		App. C					
<u>Documentation</u>							
Sys Spec	App. A,B		App I, 10	App. III, 30			
S/W Rev Plan							
S/W QA Plan					5.9		
Prog Mgt Plan							
Integ Log Plan							
Support S/W Plan						3.1	
<u>Configuration Management</u>							
CM/CSA Plan							
<u>Test and Evaluation</u>							
Test and Evaluation Plan		10.4					

**Table 8.2. Correlation of Standards and Ideal Cycle (2 of 4)**

**B. Specification and Design Phase Sections**

<u>Rev./Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<u>Revised</u> <u>52779</u>	<u>480</u>
Sys Design (SDR)	20.				3.2.6	
Prel Design (PDR)	30.				3.2.6	
<b><u>Documentation</u></b>						
Prog Spec	20.	60.	60.4	5.1	3.2.2,3.2.4, 3.2.8	
Prog Design Spec	40.	130.	60.5	5.2	3.2.2,3.2.4	
Interface Design Spec	10.		20.		5.12.4b, 5.12.5c, 5.2.3	
Data Base Document	30.				5.2.2.6	
<b><u>Configuration Management</u></b>						
CM/CSA			140.	5.11	3.2.7	
Software Errors				5.10.3	3.2.9	
Software Patches				5.10.3.2		
<b><u>Test and Evaluation</u></b>						
Subprogram Tests		30.4		5.10.2	3.2.8	
Function Tests		60.4			3.2.8	

Table 8-2. Correlation of Standards and Ideal Cycle (3 of 4)

<u>C. Development Phase Sections</u>						
<u>Rev./Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<u>Revised 52779</u>	<u>480</u>
Critical Design (CDR)	3.4 40			4.4 5.12.3	3.2.6	
Function Configuration Audit (FCA)	3.5 50			4.4 5.12.3	3.2.6	
Physical Configuration Audit (PCA)	3.6 60			4.4 5.12.3	3.2.6	
<u>Documentation</u>	<u>60.1</u>					
Program Design Doc (C5=Part II, etc)	40.1.3.1(a) 40.1.3.2(a)	130	60.1,60.2 60.2.1(b) 60.5	3.6.2,5.2 5.12.3	3.2	
Program Packages (program on deliverable media, in source and object, plus listings, X-refs, etc.)	40.1.3.2	4.5	60.5.5 60.5.4.2	5.5.6 5.5.7, 5.6, 5.7 5.12.3.1(d)	5.0	
<u>Configuration Management</u>						
CM/CSA		3.1.1	Entire document	4.5,5.11	3.2.7	
Software Errors				3.9.1, 5.8.5, 5.10, esp 5.10.3.1		
Software Patches				3.15,5.10, 3.2		
<u>Test and Evaluation</u>						
Performance Tests	3.5 50	60.4	60.4.4 60.5.4	5.8.3, 5.10	3.2.8	

Table 8-2. Correlation of Standards and Ideal Cycle (4 of 4)

<u>D. Evaluation Phase Sections</u>					
	After Product Baseline - Before Operational Baseline				
<u>Rev./Audits</u>	<u>1521</u>	<u>490</u>	<u>483</u>	<u>1679</u>	<u>Revised</u> <u>52779</u>
<b>Formal Qualification Review (FQR)</b>	App. G	4.3.2 4.4 App. VI App. XIII	App. VI	5.11.1.2	3.2.8
<b>Documentation</b>					
Operator's Manuals	App. G	App. VI	App. VI	4.4	
User's manuals		60.5.4.1	60.4.4	5.8	3.2.4
Test Reports		App. XIII	60.5.4.1	5.9	3.2.8
<b>Configuration Management</b>					
CM/CSA					
Software Errors					
Software Patches					9.1
<b>Test &amp; Evaluation</b>					
Integration Test					
System Integration Test		4.4.1.2 4.4.2	4.4.1.2 4.4.2		
Acceptance Test		4.6.4	4.6.4		
Tech. Eval. Op. Eval.		App. VI	App. VI		

- a. 3 = primary approval point
- b. 2 = basis for activity but not primary approval
- c. 1 = check for status change
- d. 0 = status check only.

An example of how this table is to be used can be gained by looking at two different products. The system level specification (Table 8-3) is the first software product to be developed, and the primary approval/disapproval events occur at system requirements review (SRR) and again at systems design review (SDR). Thereafter, it is important to establish that the requirements have been properly implemented in the design (values of two at preliminary and critical design reviews) and fully implemented (value of two at formal qualification review). During the functional configuration and product configuration audits, a value of one is assigned to indicate that status changes are looked for.

By contrast, the computer program package (CPCI) will not have been developed to the state that acceptance criteria can be applied until the two audits. The FCA and PCA then become the two events at which primary approval of the code is obtained.

Table 8-3. System Level Specification Set (FCI) Acceptance Criteria

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Are s/w functions adequately defined for the system?	3	3	2	2	1	1	2
2. Is a test requirement adequately defined for the system?	3	3	2	2	1	1	2
3. Are s/w functional requirements consistent with user requirements?	3	3	2	2	1	1	2
4. Is the FCI in agreement with applicable documentation standards?	3	3	2	2	1	1	2
5. Are s/w functional requirements consistent with interfaces?	3	3	2	2	1	1	2

**Table 8-4. Software Acceptance Criteria for Development Spec and Program Requirements Spec (1 of 2)**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Software requirements are mapped from system level specs into software requirements specs.	0	3	2	2	2	1	1
2. Mapping consistent, complete, accurate. Vertical and horizontal consistency and compatibility achieved.	0	3	2	2	2	1	1
3. Each functional requirement is defined explicitly, quantitatively and testably in terms of inputs, processing, outputs, data requirements, interfaces, accuracy, timing, exception handling, constraints, and pertinent performance.	0	3	2	2	2	1	1
4. Maintainability, usability, and reliability requirements are sufficiently well defined.							
a. modularity	0	3	2	2	2	1	1
b. structuredness	0	3	2	2	2	1	1
c. descriptiveness	0	3	2	2	2	1	1
d. consistency	0	3	2	2	2	1	1
e. simplicity	0	3	2	2	2	1	1
f. expandability	0	3	2	2	2	1	1
g. testability	0	3	2	2	2	1	1
h. device independence	0	3	2	2	2	1	1
i. self-containedness	0	3	2	2	2	1	1
j. robustness/integrity (resistance to noise)	0	3	2	2	2	1	1
k. accessibility	0	3	2	2	1	1	1
5. Are the data base and date requirements clearly stated?	0	3	2	2	2	1	1
6. One-to-one mapping of requirements (Section 3 of spec) into a sufficient method of evaluation (Section 4 of spec).	0	3	2	2	2	1	1
7. Are the requirements for software structure, etc., clearly stated?	0	3	2	2	2	1	1

AD-A103 485

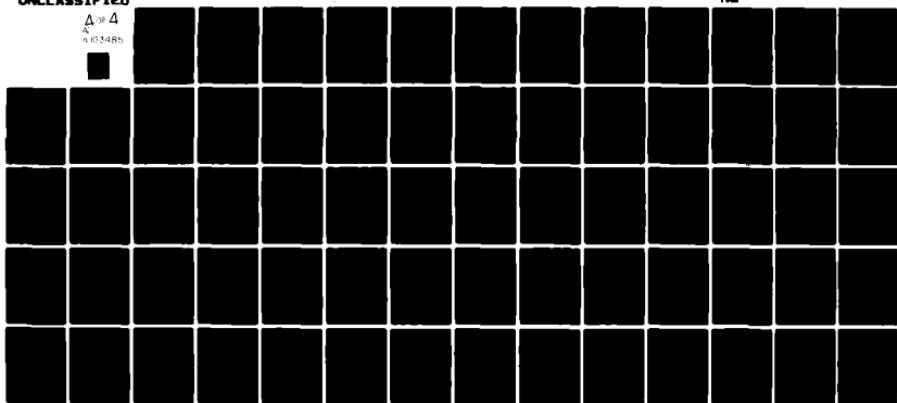
AIR FORCE LOGISTICS COMMAND WRIGHT-PATTERSON AFB OH  
PROCEEDINGS OF THE JOINT LOGISTICS COMMANDERS JOINT POLICY COOR--ETC(IU)  
AUG 79

F/G 5/1

UNCLASSIFIED

Delta 4  
4  
6103485

NL



END  
DATE  
FILED  
40-811  
DTIC

**Table 8-4. Software Acceptance Criteria for Development Spec and Program Requirements Spec (2 of 2)**

		SRR	SDR	PDR	CDR	FCA	PCA	FQR
8.	Conformance with system, accuracy control, and interface control specifications (i.e., other equipments, operators, other software and data/data bases).	0	3	2	2	2	1	1
9.	Are the equations scientifically correct and consistent with the requirements?	0	3	2	2	2	1	1
10.	Reasonably achievable (feasible) within allocated resources and schedules.	0	3	2	2	2	1	1
11.	Are requirements for system resource margins adequately specified?	0	3	2	2	2	1	1
12.	Acceptable risk.	0	3	2	2	2	1	1
13.	The specification clearly distinguishes between each requirement and information that does not constitute a requirement. (All "shall" statements should be requirements.)	0	3	2	2	2	1	1
14.	Error processing logic must describe CPCI performance when improper, incorrect, or out-of-range inputs are received.	0	3	2	2	2	1	1
15.	All system limits and capacities are compatible with the System Specification.	0	3	2	2	2	1	1
16.	Conformance to documentation standards.	0	3	2	2	2	1	1

**Table 8-5. Software Acceptance Criteria for Design Documentation (Product Specs) (1 of 2)**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Software module structure, interface, and layout conform to requirements specs.	0	0	3	3	1	2	1
2. Memory estimates, computations speeds, I/O budgeting and resource utilization are reasonable and consistent/compatible with interface and accuracy control specs, hardware/system specs, and resource allocations.	0	0	3	3	1	2	1
3. Accurate, consistent, complete translation of requirements specs (development spec).	0	0	3	3	1	2	1
4. Software explicitly described in terms of:							
a. Algorithms/equations	0	0	3	3	1	2	1
b. Narrative form, logic, timing diagrams, memory/timing/I/O budgets.	0	0	0	3	1	2	1
5. Satisfies maintainability, usability, and reliability requirements.	0	0	3	3	1	2	1
6. Algorithms satisfy accuracy, interface, timing, and response requirements.	0	0	0	3	1	2	1
7. All software requirements have been addressed in the design and there is traceability.	0	0	3	3	1	2	1
8. The data base is fully defined and its architecture (structure and access methods) are fully compatible with the logical design.	0	0	3	3	1	2	1
9. The specific module capabilities are defined.	0	0	3	3	1	2	1

**Table 8-5. Software Acceptance Criteria for Design Documentation (Product Specs) (2 of 2)**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
10. Control structures and data linkages are consistently defined.	0	0	3	3	1	2	1
11. Sufficient timing and sizing margins exist.	0	0	3	3	1	2	1
12. The design is detailed enough to begin coding.	0	0	0	3	1	2	1
13. The design is achievable within allocated resources, schedules, and acceptable risks.	0	0	3	3	1	2	1
14. Conformance to documentation standards.	0	0	3	3	1	2	1

**Table 8-6. Software Acceptance Criteria (SAC) for the CPC<sub>I</sub> (CODE) (1 of 2)**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Software is coded in accordance with the design (product) spec.	0	0	0	0	3	3	1
2. Documentation in the code is complete, descriptive, consistent, and conforms to programming and military standards.	0	0	0	0	3	3	1
3. The code produces correct output for prescribed inputs:	0	0	0	0	3	3	1
a. Arithmetic results are correct for nominal conditions.							
b. Minimum and maximum inputs are processed correctly.							
c. Singularities and other conditional occurrences of data are processed correctly.							
4. Subroutine calls are properly formulated.	0	0	0	0	3	3	1
5. Parameters are dimensionally correct, and invoked in proper calling sequences.	0	0	0	0	3	3	1
6. Scaling is proper to realize correct precision and desired results.	0	0	0	0	3	3	1
7. All error conditions have been properly processed.	0	0	0	0	3	3	1
8. Timing and resource allocations have been properly mechanized.	0	0	0	0	3	3	1
9. Task sequencing is proper to mechanize the function in correct execution order.	0	0	0	0	3	3	1
10. Software shows modularity and simplicity.	0	0	0	0	3	3	1
11. The purpose of the code, and the source coding used, are fully described.	0	0	0	0	3	3	1

**Table 8-6. Software Acceptance Criteria (SAC) for the CPC1  
(CODE) (2 of 2)**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
12. Source code and its descriptions are self- (and mutually) consistent and uniform.	0	0	0	0	3	3	1
13. Interfaces between routines, and to external devices, are well designed and described.	0	0	0	0	3	3	1
14. Source code is concise and produces efficient, relocatable machine code.	0	0	0	0	3	3	1
15. Source code is maintainable, usable and reliable, in terms of:	0	0	0	0	3	3	1
a. Structure							
b. Consistency							
c. Expandability/modifiability							
d. Testability							
e. Device independence							
f. Self-containment							
g. Robustness/integrity (resistance to noise)							
h. Accessibility.							
16. Data base structure implemented in code is self-consistent and hierarchical.	0	0	0	0	3	3	1

**Table 8-7. SAC for Test Plan Documents**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Requirements traceable between requirements specs and test plans.	0	3	3	2	2	1	1
2. Resources and facilities are adequate and consistent.	0	3	3	2	2	1	1
3. Test schedules are feasible and consistent.	0	3	3	2	2	1	1
4. Organizational responsibility is clear.	0	3	3	2	2	1	1
5. Test objectives are clear.	0	3	3	2	2	1	1
6. Acceptance criteria are completely traceable.	0	3	3	2	2	1	1
7. Test method type is traceable back to Section 4 of requirements document.	0	3	3	2	2	1	1
8. Test cases are efficient and complete.	0	3	3	2	2	1	1
9. Test dependencies are adequately defined (tests which are the basis of later tests should be done first, etc.)	0	3	3	2	2	1	1

**Table 8-8. Software Acceptance Criteria for Test Procedures**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Pass/fail acceptance criteria are based upon expected values properly derived from test requirements.	0	0	0	3	3	1	1
2. The scope, objectives, and unit under test are clearly stated.	0	0	0	3	3	1	1
3. Explicit, accurate implementation of the test plan is provided.	0	0	0	3	3	1	1
4. The test script is clear, traceable to test plans and requirements, consistent with software requirements and design, and exhibits adequate coverage.	0	0	0	3	3	1	1
5. Test data requirements and data reduction techniques are complete and traceable to acceptance criteria.	0	0	0	3	3	1	1
6. Acceptance criteria are complete and traceable to test plans.	0	0	0	3	3	1	1
7. Documentation standards are followed correctly.	0	0	0	3	3	1	1

**Table 8-9. Software Test Reports' Software Acceptance Criteria**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Software is tested in accordance with approved test procedures.	0	0	0	0	3	1	1
2. Test results meet the minimum requirements stated in development specifications and validate performance.	0	0	0	0	3	1	1
a. A sufficient number of test cases are run.	0	0	0	0	3	1	1
b. Boundary conditions are recoverable.	0	0	0	0	3	1	1
c. Adequate analysis has been performed and proper interpretation and evaluation has been made.	0	0	0	0	3	1	1
d. Test results conform with requirements (development) specifications.	0	0	0	0	3	1	1
3. All errors have been identified and evaluated and corrective action was identified.	0	0	0	0	3	1	1
4. Deficiencies were identified and evaluated and corrective action was identified.	0	0	0	0	3	1	1
5. Waivers and/or deviations were identified and documented, and sufficient rationale is available for evaluation.	0	0	0	0	3	1	1
6. Documentation standards are conformed to.	0	0	0	0	3	1	1

**Table 8-10. Software Acceptance Criteria for Operators' and Users' Manuals**

	SRR	SDR	PDR	CDR	FCA	PCA	FQR
1. Complete description of user/operator options, controls, and functions.	0	0	0	0	0	3	1
2. Safety limitations are defined.	0	0	0	0	0	3	1
3. Manuals are consistent with software design.	0	0	0	0	0	3	1
4. Manuals are adequate to support training.	0	0	0	0	0	3	1
5. Operator Interaction sequences of events are completely and clearly defined.	0	0	0	0	0	3	1
6. Recovery techniques are adequately defined.	0	0	0	0	0	3	1
7. Operational limits are described.	0	0	0	0	0	3	1

## ACTION PROCEDURES

Upon detection of a failure of the software to satisfy acceptance criteria, the software manager must determine the appropriate action to be taken. The actions available range from an unqualified refusal to permit the developer to proceed to the next development step, to electing to ignore or waive the failure. The technical/management decision that is made can have a far-reaching impact upon contractual relations and total system acquisition. Herafter a guide is presented that gives the software managers a recommended action based upon three criteria. These are:

1. What is the impact to the system acquisition of the criterion failure (or error severity level)?
2. What is the cost to correct the criterion failure?
3. What is the impact to the system acquisition schedule or cost, if the failure is not corrected?

### 1. Determination of Error Severity Level

The software manager should analyze the acceptance to determine its potential impact on the proper functioning of the embedded computer system functioning. Once this analysis is complete, the criterion failure shall be documented into one of four categories of severity. These four categories are defined as follows:

- a. Severity 1: "Prevents accomplishment of its primary function, jeopardizes safety, or inhibits maintainability of the software"
- b. Severity 2: "Degrades performance or maintainability, with no workaround"
- c. Severity 3: "Degrades performance or maintainability, but a workaround exists"
- d. Severity 4: Does not adversely affect performance or maintainability" (such as documentation and similar errors transparent to users).

### 2. Estimation of Cost to Correct

After the severity of the failure has been categorized, the cost of correcting the failure should be estimated. Parameters that are important in estimating the cost are:

- a. Manhours of labor
- b. Equipment costs
- c. Test costs
- d. Documentation
- e. Review.

The software manager should then define at least three cost categories for the particular acquisition. One way of defining these relative cost categories is the cost of making the correction as a percentage of the total software development cost. One such categorization could be:

<u>Category</u>	<u>Cost (percentage of software development cost)</u>
Low	Less than 0.5 percent
Medium	0.5 to 2 percent
High	Greater than 2 percent

A portion of the failure analysis to be conducted by the software manager is to estimate and document the cost of correcting the error and ascertain the appropriate cost category.

### 3. Determination of Acquisition Impact

The third parameter to be determined by the software manager is the impact that holding development of the software until the failure is corrected will have on the system being acquired. Three categories of system impact are defined as follows:

- a. A failure requiring an "emergency" fix is an error which, if not fixed, imposes rapid increases in cost, could cause safety considerations, or causes ever greater schedule slippages.
- b. A failure requiring an "urgent" fix is one which could cause moderate increases in cost or moderate but perpetually increasing schedule impacts (delays) the longer it remains uncorrected.
- c. A failure is classified as "routine" if it causes no particular cost or schedule slippage no matter how long it remains uncorrected.

The manager's third step is to classify the impact upon the system and categorize it. This determination should be documented and concurrence of the impact given by the system acquisition program manager.

### 4. Determination of Action

Once the three action parameters are determined according to the above steps, the software program manager should determine what action should be taken about the criterion failure. The permissible steps are:

- a. "Rework it right now" (fix it immediately, even if this requires that other development or progress be halted for the moment)
- b. "Correct the error while continuing the development activity"
- c. "Proceed with no action" ("live" with the error, with a note to correct later if feasible)

The manager can determine which action is appropriate by executing the following two steps. First, he determines an intermediate factor that correlates the failure severity level and the cost of correcting the failure. This is done by a tabular lookup from the following table:

Table 8-11. Failure Severity/Cost Matrix

Cost of Correcting Error	Failure Severity Levels			
	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>
High	$\gamma$	$\gamma$	$\alpha$	$\alpha$
Medium	$\gamma$	$\gamma$	$\beta$	$\alpha$
Low	$\delta$	$\delta$	$\delta$	$\gamma$

Second, the intermediate factor is correlated with the impact to the system of not correcting the error. This is done by performing a tabular lookup from the following table:

Table 8-12. Acquisition Impact/Intermediate Factor Matrix

System Acquisition Impact	Intermediate Factor			
	<u><math>\alpha</math></u>	<u><math>\beta</math></u>	<u><math>\gamma</math></u>	<u><math>\delta</math></u>
Emergency (High schedule/cost)	L	L	R	R
Urgent (Medium schedule/cost)	L	C	C	R
Low (Low schedule/cost)	L	C	C	C

By performing these two table look-ups, a recommended action is derived as follows:

- a. R = Rework failure immediately even if it requires holding other progress
- b. C = Correct error but do not hold program
- c. L = Proceed, no action (accept the failure now with future action to correct it)

An example of how this works can be determined by considering a failure of severity level 3, of high cost, with a medium system impact. Looking into Table 8-11, an intermediate factor of  $\alpha$  is derived. Using  $\alpha$  and medium system (urgent) impact, a lookup in Table 8-12 gives a recommended action of L or "Proceed with no action". The software manager should not document this recommended action as the final result of his analysis. His final step is to review this recommendation and determine if other circumstances exist that might alter his recommendation. If not, he may proceed into the next development phase. If circumstances do exist to alter his conclusion, he should document this conclusion and proceed accordingly.

APPENDIX 9 - ADDENDUM

## An Addendum Report for the Software Acceptance Criteria Panel

The attachments are included as a "minority report" only in the sense that they were finished just prior to the June 20, 1979 review meeting, and have not been reviewed by other panel members for concurrence or suggested changes. The statements thus might or might not reflect the views of other panel members on any particular point.

Attachment I is a proposed method of "quantizing" whatever software acceptance criteria are ultimately decided upon. It is based upon an already existing method developed by Air Force Test and Evaluation Center (AFTEC) at Kirtland AFB, New Mexico, and elaborated by BDM Corporation under contract to AFTEC. Some suggestions for changes or uses of this method are included in Attachment I.

Attachment II is a possible detailed set of software acceptance criteria called from various sources, with special attention to the viewpoints of panel members and the Software Acceptance Criteria Panel Report. Again, however, it has not yet been reviewed by other panel members for their concurrence or suggested changes. It begins with a comparison of three different published software quality "trees", and tries to combine or adapt these into a fourth "consensus" software quality tree. On this basis, detailed software acceptance criteria are proposed for the factors that are assumed to comprise "maintainability" criteria. The other aspect of "usability" was not addressed in Attachment II. The acceptance criteria are somewhat "loaded" towards the end of actual CPCI implementation, and the acceptance criteria for documentation are phrased in terms of their support of testability; this was a somewhat arbitrary distinction. Nonetheless, Attachment II is an effort toward detailed acceptance criteria for at least maintainability aspects of software.

Attachment III contains a rather random collection of possible clarifications or changes that might be made in Mil Std 1679 a basis for software acceptance criteria and to a minor extent in Mil Std 483/490, which was not deemed nearly as useful. Ultimately, the intent was to have all acceptance criteria traceable to Mil Std 1679 (modified), so that the consensus acceptance criteria would help drive Mil Std 1679, which would help drive in turn the acceptance criteria, and so on in a recursive fashion. This method would provide the best possible correlation between the acceptance criteria and the Mil Stds and other documentation upon which such criteria must be based.

One statement in which all panel members undoubtedly concur is our joint appreciation for the opportunity to participate and to provide input into the software acceptance process which the JLC has thoughtfully provided for us.

P. Reimann

**Attachment I:**

**Proposed Quantized Software Acceptance Criteria (SAC) for Embedded Quantized Computer Systems**

**Some Desirable Attributes of Software Acceptance Criteria**

Acceptance of software either does take place (albeit sometimes after some required changes or improvements), or does not. In either case, understandable statements should exist in written form detailing why the software in its present form is not (yet) acceptable, and what should be done to correct the stated deficiency, or else written statements should affirm its acceptability. "Software" is here taken to include both an embedded computer program that is part of a weapons system or subsystem, and the documentation and other products external to the embedded computer program's source and object language, which helps define and explain it, to fulfill any contractual obligations. However, it is convenient to separate the computer program package itself from the descriptive documentation which publishes requirements contractually specified for the software, and how the software fulfills those requirements.

Historically, reviews and audits, measured against military standards or other contractually specified standards, have determined the acceptability of the software documentation and explanations of program operation embedded within the program source, while Cat.I/Cat.II (subsystem and system tests, etc.) have determined whether the software performs acceptably.

These are all valid methods of determining the acceptability of software and its documentation. The military standards have formerly shied away from establishing any one computer architecture, programming language, programming standards or conventions, etc., so the criteria for acceptance in the military standards tended to be rather general. This allowed the same military standard to be used for acquiring any software system in any language for any computer architecture for any purpose or type of weapons system. But by the same token, documentation and computer programs for any weapons system could rather easily meet those standards, because the stated requirements were so general and nonspecific, that almost any format and content could appear to "qualify".

More recently, interest has increased in having a certain preferred set of computer programming languages, such as JOVIAL, FORTRAN (or perhaps only one language, such as JOVIAL, in one or several versions). This has paralleled other groups, such as Automatic Test Equipment (ATE) software people, who have pushed a standard ATE language, such as ATLAS, in one or more versions. Programming conventions recently have focussed on using top-down, structured programming techniques as much as possible in the languages that facilitate the use of such techniques. This provides some of the impetus toward favoring certain programming languages over others). There has been much less effort to settle on any one computer architecture, perhaps from fear of missing out on great advantages from new computer architectures.

Mil Std 1679 (Navy, Dec 1973) departs significantly from the generalities which characterized previous military standards' acceptance criteria for weapons system software. Much more specific acceptance criteria are established in 1679, particularly in testing requirements. These requirements are so stringent that probably few weapons systems driven by embedded software could pass them, unless the software had been originally designed according to the rest of 1679's stated requirements. It is shown by 1679 that more specific criteria, requiring higher quality, can be specified without stating any particular computer architecture or even programming language (except for restricting the possibilities to those approved for DOD use).

Overall, 1679's stated requirements are an improvement over the more general (vague) statements (or on some subjects, lack of statements) found in Mil Stds 490 and 483, for example.

What would be a consensus of desirable attributes of software acceptance criteria? Most would agree with the following partial list:

1. High standards for SAC tend to result in higher quality software (including documentation) during development, even before all reviews, audits, and tests determine their acceptability.
2. The initial cost of this higher quality software may be higher, but to a certain degree (of cost-effectiveness and time-schedule constraints), these extra initial costs can be recouped or more than recouped in lower software maintenance costs during the system's life cycle, especially for long life cycle items.
3. A universal set of SAC is needed, but unique weapons systems may have unique requirements or attributes, which may not exactly fit those "universal" criteria or might require some modification or supplement to such criteria, or reinterpretation of the universal SAC.

The question then becomes: can we find some specific, meaningful, universal set of SAC which applies standardized, precise measurements of software quality even between quite different embedded computer systems? Furthermore, is such a set not so unreasonably inflexible as to unduly "penalize" one weapons system in terms of meeting SAC just because its requirements are so unusual that they do not fit the usual mold?

Suppose that we agreed that no single set of "universal" SAC can ever be totally complete or absolutely perfect, but that some less than perfect set of criteria might still give a reasonable estimate of software acceptability in measurable factors. What would be the limitations of any such set of software acceptance criteria?

1. Probably all would concede that in 10 years, or 5, or 3, or less, the growing body of software experience and new developments and new policies might well require changes in the SAC. But what if we could avoid frequent changes in the SAC by merely changing the percentage of fulfillment required for

certain factors measuring quality in the SAC yardstick? In other words, if the SAC could state what a 100 percent perfect hypothetical weapons system software system would do in terms of performance testability, maintainability, etc., and subfactors of these, we might simply raise our expected "acceptability" threshold from, say, 63 percent of "ideal, perfect" software for an older contracted software system, to, say, 74 percent of "perfect" for a newer system to be contracted. Put another way, if the SAC yardstick remained relatively constant, but the number of inches up the yardstick which was contractually required (before the contract was ever let) could be raised in appropriate areas due to newly available "breakthroughs", or lowered for low-cost, minor systems, then the needed flexibility of SAC would exist, and we could fairly compare one system against even an "unrelated" system for relative software quality.

2. Any acceptance of a system's software depends in part upon the evaluator(s)' knowledge, biases, experience, and attitudes at home. And no set of SAC, whether "universal" or not, can provide perfect evaluators. So SAC in practice are no better than the people who apply the SAC. But what if the SAC inherently provided the way to judge the bias and knowledge and consistency of each evaluator himself, to place, for that particular SAC evaluation, a "confidence level" in his judgment? Then a quantitative measure of quality could begin to be believable, by measuring the extent of the "apples vs oranges" problem in each evaluator. Wouldn't this allay many complaints about fair judgments?

Perhaps, then, it is possible to partially compensate for the usual SAC problems of constantly changing criteria and fair applications by evaluators of the SAC (who will watch the watchers?). But any particular SAC would have to demonstrate that it could go far to overcome such usual liabilities. On the other hand, what are the desirable attributes of SAC?

One desirable attribute of SAC would be a set of criteria which are fully specified before the contract is bid on. In that way, potential contractors could better estimate how much it would cost to achieve the required level of quality, if the SAC were adequate to supply sufficient detail. But in the past, where very specific acceptance criteria have been stated before bids, it has often appeared, rightly or wrongly, that these were aimed to favor one particular contractor, and exclude others in competitive bidding. However, if a particular SAC set could be settled on and agreed upon as impartial once, it could then be used many times for many different systems, before bids, by merely changing the "percent of ideal" quality required on that yardstick. It might also alleviate many suspicions that a particular weapons system run by embedded computers was specified to favor one contractor, if the SAC specified before the tests was acknowledged to be impartial and universal. If test results are suspect in those cases where acceptance criteria were not established before the tests began (or were changed during the tests), by the same reasoning, then, SAC also should be precisely stated before letting contracts. Then, there is less pressure downstream to grant acceptance just to

stay on schedule, or because the software seems to roughly correspond to vague or general standards.

Another desirable attribute of SAC would be its ability to give more priority to some acceptance criteria than to others. For a given criterion, a very good or very poor compliance with the criterion should show up in the measurement as reflecting the degree of quality (not merely a "yes or no") on that point. In other words, we should have not just an idea of pass-or-fail on each point, but how well it passed or how poorly it fared, so that patterns can be seen. This requires something on the order of multiple choice, or scaled, responses. But the usual SAC used in reviews and audits, for example, is only a checklist of items, with no criteria of any stated priority any higher or lower than any other, nor anything more than a "pass" or "fail" response. Certainly, any criterion must be measured ultimately in the "pass" or "fail" category, but the degree of that rating and its relative priority, are too important to be lost by equally weighted yes-no responses. One favorable attribute of weighted criteria is the ease of simply changing the relative weights of the priorities of each criterion as different policies or technical developments emerge, without necessarily having to change the statement of the criteria so frequently. Thus, different expectations can readily change from one system to another, but this is done by adjusting priorities and expected degrees of perfection by shifting the points on the yardstick, not changing the yardstick itself. The standards or criteria should be as durable as possible, as an always-changing yardstick makes for several problems in continuity of measurement.

Anything can be overdone, whether measurement, quality, or whatever, in terms of the benefits derived versus the cost. Suppose, to take a hypothetical example, System A, with embedded computers, was developed to seek out and mark with Red Dye Number 2, livestock in enemy territory. But System B involved arming, aiming, and releasing nuclear weapons from a new aircraft. Clearly, the quality of System B's software must be as close to 100 percent as it is possible to achieve, though its initial cost will be higher. But System B might only have to have a quality level of 55 percent of ideal, as reliability in marking "enemy" cattle matters less, and occasionally marking a herdsman by mistake is relatively harmless. But if each system is in many respects a relatively "new" or unique system, how can we tell beforehand how much quality (what percentage of 100 percent "perfect" software) we can reasonably pay for? If the SAC changes with each system, we have to guess with each system. But if the same SAC is used between many systems, we can, if this SAC is quantifiable, produce from that experience a "family of curves" of estimated cost (in constant year dollars, for which another curve can provide now-year cost adjustments) versus percentages of "quality" attained on various points of the relatively fixed yardstick. That would increase the certainty of estimates about how much a given quality level would cost for a given required system. It would also facilitate comparisons of the past performances of different vendors for given costs, which might be a consideration in awarding future contracts, and would provide incentives to contractors to deliver "the best for the buck." A quality level for a given predicted cost is language that anyone can understand, though it is not so easily

arrived at. But programs have to be sold to Congress after being sold to the President, and cost versus quality is an important consideration in choosing systems.

As the converse of the limitations of bias and experience, it is desirable for an SAC set to include a meaningful "confidence level" indication of the evaluators' own expertise and care. This means, for best results, statistical analysis of the evaluators' scaled (i.e., multiple choice answers) or quantitative answers to each criterion. Self-consistency checks, done by computerized analysis, could detect patterns that were self-contradictory by individual evaluators, or reveal individual biases on certain topics by comparison with other evaluators. Weights might be given also to the experience of the evaluator and his knowledge of the system being evaluated. Evidence of "collaboration" between individual evaluators (which undermines the independence of each judgment, and the statistical significance of the collective judgment of the individual responses) might also be detectable by patterns of identical or near identical responses on each question. Evaluators are only human, and some measure of the dimension of depth of significance and self-consistency and bias needs to be made of evaluations in order to retain objectivity in the evaluation. Once a measurement individual bias, experience level on this system, self-consistency or care in responses, etc., has been made for each evaluator of this system by his response patterns, these can be compensated for statistically. This "purified" or compensation set of results would have a much greater confidence level. This would have to be done by a separate data collection and scoring agency that has set up such methods of statistical analysis for any SAC set, for any software driven weapons systems. It would only work for SAC quantized responses, which can be quickly and unerringly coded for computerized analysis.

While a limited set (i.e., multiple choice or scaled responses) of answers to each criterion is necessary for statistical evaluation and measures of groups of "percent of ideal" achievements of software attributes, it is also desirable to have subjective, essay-type responses detailing problems or advantages of the software. These of course are not amenable to statistical scoring, but no set of SAC, however universal it might be, can completely reduce all problems or successes of a software system to ones and zeros. The SAC set thus must not exclude essay responses, but in fact should elicit and encourage them. A quantized SAC set is thus never self-sufficient as a means for acceptance or rejection of any software, but is complementary to essay-type evaluations such as might result from reviews and audits, etc. Neither can it replace such subjective or essay type critiques, but it can add another dimension of "percent of quality" for various standard factors, which is easily understood by decisionmakers who are not all familiar with the details of this particular weapons system.

A good set of SAC will have explicit statements of what is to be measured or evaluated for each criterion, and does not leave these to be merely inferred, which will cause larger variance of responses between different evaluators.

A good set of SAC will also have a carefully organized structural hierarchy showing the relationship of different criteria to each other in as nonredundant a form as possible, except where needed for self-consistency checks or broad overviews.

Any proposed SAC should correlate as closely as possible to present Mil-Std provisions, or proposed revisions to those Mil Stds. Otherwise, the SAC is more likely to be in conflict with the Mil Stds the less it correlates with them. This makes such SAC, however, "valuable", unenforceable. So, SAC should closely follow applicable Mil Stds, and at least not be in conflict with them or proposed revisions to them.

But in developing any hierarchical SAC that develops certain factors, and define and bound the relationships between different factors, some possible improvements to the Mil Stds will probably become evident from the act of organizing a hierarchical SAC structure. Thus, the SAC should not merely be derived largely from the Mil Stds, but should also bring out possible improvements in the applicable Mil Stds. This makes the SAC and the Mil Stds a recursive pair, each driving the other in turn. This close relationship is necessary to justify those SAC and make them less arbitrary, for if not grounded in the Mil Std provisions (for which we might be able to suggest improvement, but hardly a total rewrite in our limited time), what are the grounds of justification of the SAC? That is, on what grounds can we defend the SAC as non-arbitrary, if not by the Mil Stds? Only a set of SAC based largely on applicable Mil Stds has much chance of adoption, for otherwise it will be rejected for being in violation of the Mil Stds, or at least for being impossible to correlate with them. The SAC set should provide a convenient way to measure (or quantize) conformance to particular Mil Std paragraphs or provisions.

If the methodology underlying any set of SAC has any pretense of being suitable for "universal" applications, the working of the SAC should be self-evident merely from filling out one such SAC set as an illustrative example.

If the results of SAC formulation and measurement must be understood by far more people than the number of people doing the formulation of relative weights and measuring the perceived results, it is more important for the SAC to produce a brief, easily understood summary result than to provide for easy evaluations. Far more readers would be capable of, and more important, willing to read a summary like "this software system attains an estimated 74 percent of ideal maintainability, just over the contractually required 71 percent", than a mountain of technical essays. Such concise statements can lead into more detail of what maintainability deficiencies exist, but unless the SAC set lends itself to concise summaries, most readers won't even look at the details that follow. But even if it is less important to evaluate than to concisely summarize, the evaluation method should be essentially the same between systems so as to facilitate comparisons between different systems.

The results of evaluation are nearly as "repeatable" (reproducible) as possible. That is, the responses of an evaluation team, collected together due to statistical checks, etc., are nearly the same, whether done on Monday or Friday, or by Team A or by Team B. To the extent that this is true, the evaluation can be said to be truly objective. Clearly, essay-type responses or evaluations will probably vary more from day to day or between teams than multiple-choice scaled responses might vary.

Cost considerations should allow a less exhaustive or less detailed evalaution for lower cost systems than for high cost, longer lived systems, whether the evaluation is essay type statements, or multiple choice, scaled responses. Fewer evaluators should be required for less costly or less critical systems (at the cost of less "confidence" in results), so that cost-effectiveness measurements do not excessively warp the costs they are trying to measure. A good set of SAC should likewise make it easy to express lower required acceptance standards as well, without having to rewrite the SAC altogether.

#### Weighted Multiple Choice (Scaled) Responses

Varoius "checklists" have long been used to assure that certain stated software concerns are not omitted or forgotten at various software events. To use Air Force standards and guidebooks as examples, Section 130, Figure 18, of Mil Std 483 contains a checklist of considerations for Engineering Change Proposals, called an Engineering Change Classification Checklist. This is not precisely a quality measurement checklist, but is similar to one. The Air Force Guidebook on Verification has numerous sections that amount to checklists of items that a Software Director should check when various contractor documents are delivered. The AF Guidebook for Reviews and Audits contains explicit sample checklists (of the "yes/no" variety) for PCAs and FCAs, etc. Various matrix analyses (such as in the AF Cost Estimation and Measurement Guidebook) amounts to multiple choice responses (though not exclusive responses) of columns to rows. Checklists thus range from informal to formal, from single "yes/no" (two column) responses to multiple choice (various columns), scaled (single answer) to several possible joint answers.

To the extent that all systems have to satisfy such checklists of requirements in Mil Stds or Guidebooks used by SPOs and SDs, such a measurement technique has already been widely used across a broad spectrum of systems. However, for each "checklist" item, the measurement has typically not been made in terms of a percentage of ideal, achievable quality which might be reached, and seldom has even a scaled response (of multiple choices) been used. Usually, answers amount to "yes, it satisfies the requirement", or "no it does not". Ultimately, exactly such a decision (yes or no) has to be made, but if this is made immediately, and the estimation of the degree of compliance is then sacrificed, overall patterns of compliance and collective compliance become obscured, because the scale or degree of compliance of each item is lost.

Further, not all factors should carry equal weight, and a mere "yes/no" checklist approach does not necessarily recognize that some checklist factors matter more than others. Even if the SPO or SD mentally accords more importance to some one factor than another, no specific notation is made of how much more important he deems that factor than the other factor.

One novel approach of some possible value might be to consider assigning negative weights to some acceptance criteria when they may be associated with degrading quality in one aspect, while improving quality in another aspect. For example, in software, efficiency is a desirable attribute or quality in itself. But certain aspects of efficiency may detract from maintainability; for those, an evaluation of maintainability might profitably assign to those factors some negative weight in evaluating maintainability, but their usual positive weight in evaluating efficiency for its own sake. Too often, instead of using such known relationships, which include deleterious effects, they are simply avoided, with the acknowledgment that the results must therefore be discounted somewhat. But it is much better to make use of those relationships by negative weighting in some cases, instead of avoiding them and having to admit that the results must therefore be discounted somewhat. This is touched upon also as a note under level 1 of the MODIFIABILITY aspect of Attachment II.

What is ultimately desirable is both a measure of the relative merit or acceptability of all the myriad factors and subfactors that affect how well embedded software works, how usable it is, and how maintainable, and also its overall acceptability in those areas. The AF Guidebook on Software Quality (Figure 6) attributes software quality to two superfactors: functional performance, and maintainability. In turn, functional performance is assumed to be composed of factors of reliability, efficiency, and human engineering, while maintainability is assumed to consist of factors of testability, understandability, modifiability, and portability. In turn, the factor of reliability is composed of subfactors of correctness, integrity, and consistency, while efficiency is made of accountability, device efficiency, and accessibility. Human engineering is assumed to consist of subfactors of integrity, accessibility, and communicativeness. The factor of testability is to be made of factors of accountability, accessibility, communicativeness, self descriptiveness, and structure. In the Guidebook's view, understandability is made of consistency, communicativeness, self descriptiveness, structure, conciseness, and readability. The factor of modifiability is assumed to be composed of subfactors of structure, augmentability, and self-containment. Finally, portability consists of self-containment and device independence. The overlap between subfactors is graphically shown in Figure 6 of the Guidebook. Superfactors correspond to level 1, factors to level 2, and subfactors to level 3 in that figure. This guidebook came out in 1977.

In 1975, AFTEC did a somewhat simpler factor analysis, in its Software Testing Guide. But it combined such factor analysis with a set of questions, each of which dealt with some aspect of software quality. Five multiple choice answers were provided as standardized responses to each statement of software quality. These answers were "a)

strongly agree; b) agree; c) neutral; d) disagree; and e) strongly disagree". Alternatively, a respondent could reply "f) not applicable". For example, statement 4 was "the program contains a capability for tracing and displaying logical flow of control". If the evaluator strongly agreed that a full capability existed, he marked "a) strongly agree". If no such capability existed, he marked "e) strongly disagree". If somewhere in between, he marked it accordingly.

The AFTEC approach did not require each evaluator to answer the set of questions for every module in the entire software systems or subsystem being evaluated. Rather, it used a "t" statistic to estimate when evaluations converged to "data stability": that is, when he had evaluated enough modules that evaluating more did not appreciably alter his overall collective response averages for the system. AFTEC assumed that each of its 35 questions relating to subprogram (module) acceptability, answered independently for each module evaluated, and its 36 program questions (answered only once for the whole system) dealt with one or more of 7 "software quality factors". These factors were: portability, reliability, efficiency, human engineering, testability, understandability, and modifiability. In turn, these seven factors were assumed to compose three superfactors: portability was made up identically of portability; "as-is utility" was composed of reliability, efficiency, and human engineering; and "maintainability" consisted of the factors of testability, understandability, and modifiability.

The relative fractional importance of each of the seven factors was decided upon (such that the sum of weights of the seven added up to one). Then each question had a "weight," which depended upon the factor or factors it was assumed to involve, and the weight of those factors. Further, the "strongly agree" response (most favorable, expressing highest quality) gave four points to whatever fractional weight the question carried due to its assumed composition of one or more of the seven factors, while a "strongly disagree" response gave zero points to its question's factors. The other responses lay proportionally in between (three points for "agree", two for "neutral", one for "disagree"; "not applicable" had no effect either way).

Then for the factor-weighted responses, total quality indexes were computed for each of the seven factors (and in turn for the three superfactors composed of those seven factors). These indexes, which amounted to percentages of theoretical "perfect" scores (which would result if all questions were answered by "a) strongly agree") were then compared with acceptance thresholds (minimum percentages allowable) that had been decided upon prior to the evaluation. If the measured percentage of quality or acceptability was below the predetermined required percentage of quality for that factor, then that factor was deemed to have failed acceptance. A similar picture was obtained for each of the three superfactors, and in turn of the overall total made by summing all seven factors (or all three superfactors). Thus, rejection might occur on any or all levels of factors, superfactors, or overall total.

The questions evolved periodically, in part due to comments by users, who employed the questionnaire's quantized approach to OFP modules, etc. Real-time (embedded computer) evaluators tended to feel that real time, interrupt-driven embedded computer programs had unique facets not found in the usual offline batch processing programs run on general purpose computers. It was admittedly difficult to find a universal set of questions that adequately served both real-time users of embedded computer systems, and batch-processing users.

Ultimately, BDM Corporation was given an AFTEC contract, and developed a revised or new set of questions. One questionnaire dealt with documentation provided for the program; another questionnaire dealt with the source listing, answered uniquely and individually for each of the randomly chosen modules taken to be sufficient for statistical purposes. Here, however, the number of modules was predetermined for selection, rather than being based upon actual convergence based upon the sameness of answers. To illustrate the numbers and hours typically involved in such questionnaire evaluations, it was deemed necessary to evaluate, for 4 F-16 OFPs (Operation Flight Programs), 30 subroutines in the Fire Control Computer OFP, 30 in the Stores Management Set OFP, 29 in the Fire Control Radar OFP, and 26 in the Inertial Navigation Set OFP. A minimum of five evaluators was required by the AFTEC criteria for each (sub)system, and each was projected to spend an average of 82 hours answering the documentation questionnaire (once) plus the module questionnaires an average of 29 times. All of the software engineers were familiar with the systems they were evaluating: a "biodemographic" questionnaire recorded general software experience as well as each individual's experience on the system he was evaluating, for use in statistical evaluations of confidence.

The BDM version was published as the Software Maintainability Evaluator Guidelines Handbook (SMEGH), (BDM/TAC 78 687 TR, 24 November 1978). In addition to a revised or new question set, questions were organized in seven separate sections: modularity, descriptiveness, consistency, simplicity, expandability, instrumentation (test recordability), and general (recapitulative). Rather than a separate questionnaire set of embedded computer users, and a different one for batch or general purpose users, the BDM version adopted a disarmingly simple solution to this dilemma. All users were required to answer every question, but if the evaluator felt that any particular software point was "not applicable" to his system, an "a" (best possible) answer was given. The rationale was that if a given software concern or potential problem did not apply to this system, then it could not have a problem in this area, and so had the highest possible quality in that area.

This leads to a slight positive bias in results, but is well worth that, because it nonetheless provides a solution to the fact that between different systems, evaluators would bypass (mark "N/A" for not applicable) a question on one system that they would address on another system. This made it difficult to compare results between two different systems. In fact, even on the same system, two evaluators who worked side by side might even have honest differences on the applicability of a given question to that

system. One might feel that it applied, and the other, from a somewhat different viewpoint or line of reasoning, might feel that it did not. Where "not applicable" answers varied even between the responses on the same system, it undermined the weighting scheme, as an "N/A" effectively removed that question from the weighting. By requiring a judgment from each evaluator on each question, and removing the "N/A" option in favor of an "a" (best quality) answer, all weightings of questions remained intact. (Weighting was now a function of which of the seven sections the question fell in).

In all cases in which an "a" answer was given because the evaluator felt that the item was not applicable (therefore no problem at all), he was required to mark "N/A" for that question number on a separate coding form that contained comments. Because these comments were likewise keypunched for retention with his multiple choice responses, it was possible to search on "N/A" answers which had resulted in "a" answers, if that was desired. The comments provision was provided to let evaluators state their rationales for a given answer if they felt torn between two different viewpoints. As an additional multiple choice answer on a question, after answering the question with a multiple choice answer, whether he "had trouble answering this question", and/or "a written comment is transcribed on AF Form 1530, punch card transcript". If several evaluators had trouble with a particular question, the question itself might be the cause of difficulty (signalling one that needed future revisions), if that happened on many different systems. Or if evaluators had difficulty answering the question on only one system it most likely indicated ambiguities or conflicts within that system).

The answer sets were somewhat modified, as follows: "a) completely agree; b) strongly agree; c) generally agree; d) generally disagree; e) strongly disagree; and f) completely disagree", with the positive statement of quality that comprised each "question". (In addition, any question could have a second and/or third response, "i) I had difficulty answering this question; and/or j) A written comment is recorded on AF Form 1530"). The six standardized responses were notable for the absence of any allowed fence-sitting or "neutral" response. In the earlier system, if evaluators did not know immediately which type of answer was most appropriate to a given question for that particular module or overall program documentation, they tended to "play it safe" by marking the "neutral" response, rather than spending a lot of effort to resolve the dilemma in their minds. This new form tended to require evaluators to put more thought and effort in their answers, and thus to make them more meaningful.

One other, more subtle difficulty resolved by replacing the "not applicable" responses with "a" responses was the matter of evaluator independence of judgment. The statistical measures of bias and confidence depended on each evaluator's independence. If a collective answer were given to a question by all evaluators (especially those who were physically co-located), this undermined the statistical checks for bias because it was as if only one response by a singly "composite" evaluator were given. The reason for multiple evaluators was to provide independent samples or answers to allow those statistical checks. If only a joint or collective answer, or an answer by only the

individual most knowledgeable on the system resulted in a single response which was merely repeated by other evaluators, differences in perspective were lost. For example, a software engineer who had figured out for himself, or had previously asked the prime contractor the answer to an ambiguity, might, with this knowledge, give a more favorable answer than a relative "novice". But if there was a real lack of quality on an aspect, such as explanation of the computerized function, the relatively less experienced person's answer would more readily express frustration with a poor or nonexistent explanation.

When some questions could be answered simply by an "N/A", without requiring a scaled or weighted answer, evaluators tended to talk over the question between themselves much more: this trend also undermined their independence.

One difficulty, however, with an excessive concern for providing a sufficiently large number of warm bodies for evaluations to give statistical checks and a sufficiently large number of modules, each of which required a complete questionnaire set of answers, was the time factor. For the F-16 evaluation nearly a man-year of effort was required, just to fill out the questionnaire on maintainability, due to requiring a sample of an average of 29 modules each, among each of five evaluators for each system. That is a significant cost, and managers might well wonder if it would be better to spend some of that time running particular tests to find software errors. Further, while the initial use of a questionnaire takes a little "getting used to", after evaluating a large number of modules on the same system, boredom becomes a problem in getting careful answers, and particularly in getting written elaborations on coding forms. It might thus actually provide better quality answers and more confidence to avoid overkill in producing 14 pounds of answers, to reduce somewhat the number of randomly-selected modules evaluated, and the number of evaluators, particularly for smaller systems that do not have large numbers of people available and knowledgeable about them in the first place.

The latest BDM effort in refinement of the original AFTEC approach does help resolve many of the aforementioned difficulties in software evaluation for the standpoint of acceptance criteria. That particular questionnaire was (intentionally) limited to the subject of maintainability—that is, the built-in aids to allow nondevelopers of the original system to make needed changes to the software as problems arose.

AFTEC has recently produced a separate questionnaire set, in exactly the same format, and with somewhat streamlined explanations, for another area of software concern. This is the Software Operator Machine Interface Questionnaire (SOMIQ), related to software-only aspects of "usability" of the operator consoles on software systems. (This is, interestingly, almost exclusively an embedded-computer concern, much less related to batch users than, say, maintainability software acceptance criteria).

The SOMIQ addresses six areas: assurability, controllability, workload reasonability, descriptiveness, consistency, and simplicity. It is still being refined, with little or no experience on it as yet.

It seems apparent that different SAC sets are required for different superfactors, such as maintainability and usability. It is also apparent that to achieve widespread acceptability, such SAC must be more closely tied to Mil Stds and help drive them with improvements, also, as the contractual force of such SAC derives in considerable measure from the force (or lack of force in some cases) of the Mil Stds. This does not mean that a useful SAC set should be limited to Mil Std paragraphs; it should not, because any Mil Std expresses more general or overall concerns, and many SAC features are necessarily more detailed. They should be complementary and mutually supportive, not mutually exclusive, as SAC criteria.

Attachment II contains yet another set of SAC. It consists of maintainability criteria (partly derived from the AFTEC/BDM questionnaire), and testability criteria. Usability considerations are probably best addressed at present by the AFTEC questionnaire (SOMIQ). Maintainability is most thoroughly addressed here, and relies somewhat additionally on the provisions of Mil Std 1679. This is not to say that this rough draft is "better" than the BDM questionnaire on maintainability, (for example), but in Attachment II the points are organized in a more hierarchical structure, which is more in line with the paragraph number organization of Mil Std 1679, and hopefully thus more capable of interfacing with its concerns, and of exposing changes which might profitably be made in Mil Std 1679. The questions, or points, in Attachment II also consist of as many as five or six subconcerns, which, as each subconcern occupies one or more lines by itself, could be further broken down into smaller questions.

But the main reason for Attachment II is to provide an example of how the AFTEC/BDM methodology might be applied. It may be that this methodology, with the same kind of standardized answers, might be applied more broadly to any kind of software acceptance process, such as a review, an audit, or an acceptance test. The only "new" requirement would of course be the generation of an appropriate set of questions or rather statements about "quality" aspects to be satisfied by each of the applicable events or milestones. Products (e.g., documents and tapes) might also be the subject of such a methodology, subject of course to considerations of the amount of time required for each to be answered. (In this respect, the AFTEC/BDM methodology is perhaps even better applied to events and products, because the big time crunch for software evaluation, namely evaluating large sample numbers of software modules, would be reduced to a single event to evaluate, not a large number of events.)

If Panel D were then merely to recommend the use of this established and tried methodology already in some use within DOD, that might constitute much of its effective interface with the other panels, by suggesting the method for such software acceptance criteria. The other panels could, if they wished, adapt this method by providing the content on those products or events where their panels have the most expertise or cognizance of the event or product. But such a set of SAC should never be considered an exclusive one or a self-sufficient one; it should remain an optional but available tool to supplement the traditional SAC function as performed by reviews, audits, and tests, as detailed in the Mil Stds, and other documents.

A recent Air Force Inspector General's item (TIG Brief 3, 1979, p.3) noted that "we may tend to measure what is easily measurable, not necessarily what is useful". It suggests that instead of finding the easiest things to measure, we would instead "select those things that are most meaningful", by asking the following questions:

1. Who will use the 'measurement'?
2. How will it be used?
3. How important is it with respect to the total inspection?
4. Are there other things we have overlooked that are difficult to measure, but significantly more important?"

Clearly, the quantized approach is only one of many tools, but it may help measure and put a finger on software quality aspects, in terms of acceptability, that are otherwise difficult to measure. It does produce convenient, easy-to-read one-line summaries of "percent of ideal perfection" for any number of hierarchied software attributes which can be read and understood by the widest possible audience, including the President and Congress, if they so choose, at little cost of valuable time. It can never replace detailed essay itemizations of particular problems, but may help put them in perspective.

The fact that such an approach makes explicit statements about the weight or relative importance of the factors of software quality deemed most important for a particular system puts those assumptions "up front", where they can be seen (and debated). This is more easily provided before contracts are bid on, as the criteria remain relatively stable, but the assessed weights and acceptance thresholds upon which the contractor will be judged in terms of his performance are again explicitly stated in fair terms.

Such a yardstick of relative stability is in its infancy. If it is a valuable tool, its value can only be obtained by at least making its existence known and by developing various applications of this methodology.

**Attachment II: Proposed Detailed Acceptance Criteria  
For Software Maintainability**

## Detailed Software Maintainability Acceptance Criteria

### I. Software Is MAINTAINABLE

#### A. Maintainable software is UNDERSTANDABLE in structure and DESCRIPTION.

1. The software is MODULAR and has SIMPLE STRUCTURE which follows stated standards.

a. Unambiguous control structure characterizes program flow (Control always returns to the calling routine; and priorities or hierarchies are predetermined in case of competition, etc.).

b. Top-down, structured, forward-only control flow is from higher to lower routines (go-to's are scarce, and not backwards, except loops): few statements need labels.

c. Each routine does only related functional tasks, and each functional tasks is an easily recognizable block.

d. Each routine has a single entry point and a single exit point.

e. Each decision point, and each iteration loop, has a single entry point, and each has a single exit point.

f. No indirect, tangled, in-and-out, backwards, or "bird's nest" branching occurs, and no "scattergun" branches occur, to widely dispersed, ill-ordered targets.

g. No indirect, tangled, in-and-out, or excessively indexed addressing occurs, and no "scattergun" addressing occurs to scattered, ill-ordered parameters.

h. Data structures (e.g., tables) are simple, hierarchical, modular, and clear, and illuminate relationships between elements, but avoid multidimensional arrays.

i. Variables have only one purpose or use (for example, not used for bit flags and latitude); aliases are minimized but always identified, and overlays or equivalencing (memory sharing) is minimized, particularly unlike types (real/int).

j. Globals (variables or constants used by two or more routines) are minimized and are hierarchically organized to avoid data communication complexities.

k. Masking of bits, bit manipulation, machine dependencies, etc., are minimized.

- i. Interdependencies between routines are minimized to reduce complexities.
- m. Sequencing dependencies (on interrupts, etc.) are minimized and made clear.
- n. Modules do not call themselves, nor are subtle, tricky techniques used: self-modifying addresses are avoided, and self-modifying instructions are avoided which mutate with data or register values.
- o. Compound control logic (IF(NOT A. AND B. OR C.)GOTO) is avoided where possible, and all unnecessary nesting of control is avoided.
- p. Repeated code is subrouted for repeated access, not duplicated redundantly.
- q. No unreachable instructions exist: and little unneeded code exists.
- r. Routines average 50-100 executable statements or less, and none exceed 200; only one executable statement exists per line of source.
- s. There are not excessive numbers of expressions controlling branches (IF, DOWHILE, CAS); excessive numbers of operands are not used in this routine.
- t. Routines are grouped functionally, hierarchically, and logically in the source.
- u. Specification statements (formats, equates, etc.) are grouped functionally, and are not unreasonably far from their uses.
- v. Program control can be modified without modifying data structure, and data structure can be modified without modifying program control; i.e., data structure and program control are independent (vs. I.B.5.j).
- w. The CPCI has 20 percent or more spare time and 20 percent or more spares in all memory types, so that understandability has not been forcibly sacrificed for efficiency's sake.
- x. A common, approved high-order language (HOL) is used, and the source is never in microcode.
- y. No unnecessarily complicated math beyond basic algebra is used if at all possible, and parentheses and conventions help clarify compound math expressions.
- z. Alphanumerically- and location-sorted cross-references are supplied with, are part of, and are based upon side-by-side listings of source code and its resultant object code.

- (1) Each data name is referenced to the location (locally and globally) of every other statement which can access that data name.
- (2) Each statement that can be branched to or invoked is referenced by the location of all statements which can transfer control to it.
- (3) Each routine is tied to the locations in all other routines that may call it.
- (4) All cross-references are tied to the smallest compilable/assemblable routine names (subroutine VELOCX, for instance).
- (5) For machine-generated cross-references, a separate printout of the cross-reference is available which contains footnotes correcting all errors in the cross-reference, which result from addressing offsets and indexing, etc.
- (6) All external references are resolved in the cross-reference listing.
- (7) The cross-reference aids understanding of initialization/modification of all variables.
  - aa. Programming conventions and documentation follow approved specs, but these are not needlessly vendor-peculiar; instead, they follow industry standards.
  - bb. Resource allocation (storage, timing, mass storage devices, I/O channel allocations, consoles, etc.) is fixed throughout program execution.

2. The source code listing is SELF DESCRIPTIVE, and the code and its purposes are EXPLAINED.

- a. Parameters (variables and constants) are fully described as to type, access frequency and protection, scaling, range, limits, accuracy, precision, bit resolution, etc., in central parameter description blocks, and in using routines' preface comment blocks.
- b. Adequate comments always accompany data organizations, whether global or local, and fully describe data structures and hierarchies.
- c. Adequate comments explain the entire program's purpose, structure, priorities, sequencing of routines' cycles, relations between functional groupings, and initializations.
- d. Adequate comments exist at the beginning of each functional grouping of routines, and explain the functional grouping's purpose, and the interrelationship between routines (structure and interfaces), and initialization.

e. Preface comments meaningfully and usefully summarize each routine:

(1) Prefaces detail the purposes and functions of each routine.

(2) Prefaces detail the limitations and uniquenesses of each routine, e.g., interruptibility, accuracy, time available, I/O data limits, machine dependencies (e.g., shifts, formats, word sizes), algorithm peculiarities, stability problems.

(3) Prefaces detail unusual exists or terminations; e.g., for power loss and recovery or restart, error handling and error recovery, interrupt sources, types, rates, and required responses and save/restores.

(4) Prefaces suggest test input data and output results.

(5) Prefaces detail input and output data (including ranges and limits), initialization (including for testing), register uses, overlays or equivalences, etc.

(6) Prefaces outline logic sequencing of control, calls to other routines, and reasons for these.

(7) Prefaces list other routines that call this one, and under what conditions or restrictions, if any, that the calls are made.

(8) Prefaces state the routine name, revision number and date, programmer, and references to source or derivations for all algorithms used.

f. Comments throughout the body of each routine's source code illuminate it, giving the purpose and operations of each lowest level block of source code.

(1) Sufficient, but not excessively long, blocks of comments clarify each task, and intermediate results are described in stepwise fashion.

(2) Comment blocks provide "road maps" for decision points and help find targets (labels), and the purposes of all branches are made clear.

(3) Consistent and separate indentation exists for statements which outline nesting of control and hierarchical sequencing of control flow.

(4) Local comments explicitly warn of potential side effects, or widespread or large changes in operation, if certain pieces of code (statements) are changed.

g. Names or labels of parameters or program locations are self-descriptive.

(1) Names or labels of parameters (variables and constants) denote the function of each.

(2) Names or labels of parameters help identify relationships between parameters.

(3) Parameter names distinguish between global and local parameters, and distinguish by type and length (single vs. double precision, fixed vs. floating point).

(4) Parameter labels are in logical, hierarchical, understandable, easily found sequences, and are centrally located in easily found blocks.

(5) For each parameter, particularly inputs/outputs, adequate coding comments exist. These describe what that parameter represents, in terms of full description (meaning), engineering units, ranges, limits, required accuracy, precision format, content, scaling, rates, quantity if appropriate, data type, and source/destination.

(6) All data lists (arrays) are columnized for easiest readability.

(7) All source statement instruction labels are meaningful aids in finding them, and source statement labels are not used except for required branching and addressing.

(8) Modules can only access data which has been explicitly declared available for access.

h. All input/output (I/O) dependencies and peculiarities are well explained.

(1) Sequencing requirements and dependencies (e.g., on interrupts) of the I/Os are explained.

(2) All reservations of storage for buffers, I/O variables, etc., are well described.

(3) All peculiarites of external peripherals or system interfaces are well described, or cross-references point to detailed explanation of each in available documents.

i. All required data conversions, particularly for data entry and display, and all methods of accomplishment of such conversions, (analog to digital or vice versa, or base-number conversions or character transformations, for example) are explained.

3. Source statements and descriptions are (self and mutually-) CONSISTENT and uniform.

a. An explicit, uniform method of structuring and describing each routine is published and each routine follows those standards and all other imposed standards and specs.

b. All subcontracted design and coding agrees with specifications, standards, and forms imposed contractually upon the prime contractor by the Government agency.

c. All flowcharts (detailed or general) agree with and describe all source statements, and match source indentations and labels showing hierarchies and flow of control.

d. Uniform comments describe inputs, outputs, intermediate or local variables, order of arguments, purposes, and processing, and agree with and describe the source statements.

e. Functionally related data items are logically grouped and uniformly described.

f. In each routine, input/output global variables are ordered in logical groups, and understandable references point to full descriptions, if these are not fully described here.

g. In each routine, local variables are logically grouped and uniformly described, and are near and prior to statements using them.

h. In each routine, logical groupings of declarations (integer, real, data, etc.) are made, along with uniform placements of formats, error exists, comments, etc.

#### B. Maintainable Software Is MODIFIABLE.

1. (Utilize the criteria of I.A (UNDERSTANDABLE STRUCTURE AND DESCRIPTION); do not rescore.

2. Source statements are CONCISE and produce EFFICIENT relocatable object (matching) code.

Note: In some ways, efficiency and conciseness contribute to modifiability, and in other ways, some facets of efficiency detract from modifiability. Because it is known that some aspects of efficiency adversely affect modifiability (and also understandability), the factor of efficiency and its subfactors are usually avoided in any calculation of modifiability, and evaluators state that the measurements are not fully valid because measurements of efficiency conflict with measurements of modifiability and understandability, etc. But it is a mistake and a missed opportunity to avoid the interaction of efficiency with modifiability and then have to apologize for it and discount the results to boot. Rather, the measures of modifiability should include the measures of efficiency, not exclude them, and take into account the positive and negative effects of

various aspects of efficiency on overall maintainability. To do this, negative weights are assigned to those aspects of increased efficiency which adversely detract from maintainability/modifiability considerations. And we assign the usual positive weights to those aspects of efficiency which contribute to maintainability in a positive sense. Only this approach (of assigning some negative weights, in computing maintainability modifiability measures, to essentially positive accomplishments of certain aspects of efficiency) truly takes such effects into account. A separate measure of efficiency by itself, for example, can be computed by taking the absolute value of all weights (so that negative weights are made positive for the purpose of measuring efficiency by itself) to get a good measure of efficiency achieved purely in terms of efficiency.

a. Low-level coding to enhance efficiency in some small routines has not caused greater overall system inefficiencies or extra system overhead by causing excessive competition for input/output resources, nor extra saves/restores for re-entrance, nor increased burdens of interrupt queueing or processing of interrupts, I/O.

b. Implementations for greater efficiency have not created needs for more extensive program modifications than would otherwise be required when changes must be done.

c. Resource utilization measurements (e.g., execution time, numbers of I/Os) exist, and are readily accessible without code patches, etc.

\* d. The programming language utilizes optimizing compilers, etc., which can optimize object code by automatically making subtle changes in program logic.

\* e. Implementations adapt the program to different modes with little or no operator input needed, though this requires more complex software programming.

\* f. System utilities for this CPC1 (sort, search, etc.) are carefully optimized, even if this required low-level assembly or tricky programming for most-used one.

\* g. Data tables are set up for most efficient execution (operations done most often are done first, and binary splits are used) even though this may make them harder to follow.

h. Outputs have no more significant digits than inputs, and errors are minimized (rounding, truncation, etc., are done last), yet system accuracy requirements are met.

i. Mixed-mode arithmetic or needless data transformations are avoided where possible.

j. Computations are efficiently spaced (no unnecessary times, distances, etc., are done).

- k. Non-loop-dependent operations are done outside of loops.
- 3. Present resources used, limits, and present system capacities are completely described.
  - a. All architectural or configuration usages and limits are described at the start.
    - (1) Timing is used up, and limits (time reserves left) are stated at the start.
    - (2) Main computer memory is used, and limits (memory reserves left) are stated at the start of the program in the Executive, or in the data base comments.
    - (3) Usage and limits (remaining reserves) of each type of peripheral memory or mass memory are explained, especially for memory types which are not interchangeable.
    - (4) Usage and limits (e.g., remaining reserves and speed) for all buses, I/O channels, I/O controllers, external devices, etc., are commented on in the Executive or data base.
    - (5) Interrupt structure limits which could stymie growth are clearly stated in the Executive and/or interrupt handler hierarchy.
    - (6) Language or word size limits (affecting addressing ranges, etc.) are explained.
  - b. Comments explain or point to documents which describe constraints on inserting, deleting, or modifying source or object libraries (including protection features), particularly dependencies on support software idiosyncrasies.
  - c. Function capacities and limits (e.g., present number of targets simultaneously trackable) and full load constraints (e.g., extra saves/restores on interrupts if time reserves disappear) are described, both for software, and for external interfaces.
  - d. All dynamic allocations of resources (storage, timing, priorities of hardware services, etc.,) are explained in comments, which also explain how to expand storage, to what limits, whether dynamic or fixed.
- 4. Modifiable software is EXPANDABLE.
  - a. At least 20 percent of timing is still available (unused) as reserves for growth. (For high change system, 40 percent is desirable.)
  - b. Internal instruction memory reserves, already on board, are 20 percent or greater. (For high change systems, 40 percent is desirable.)

c. For each type of I/O or data base memory which is noninterchangeable with other memory carried on the system, at least 20 percent is spare (unused). Thus, the 20 percent margin must not only be satisfied for memory collectively, but individually, so that one unique type with only 5 percent spare cannot be "balanced" by another type which has 50 percent spare but which cannot replace it (e.g., ROM vs. RAM, or incompatible lengths).

d. Spare data throughput capacities (I/O channels, interfaces, buses, etc., exceed 40 percent (60 percent is desirable for high-change systems).

e. Spare interrupt levels or channels are sufficient for anticipated growth.

f. Peripherals' limitations (mass memory, displays, etc.) will not stymie future growth.

g. Word size for words used for addressing are not too small to permit reaching (higher) addresses that might be required with future program growth.

h. Physical spare room (cubic inches of space available) is sufficient to accommodate anticipated growth in all memory types or channels carried on the system.

i. All full load constraints (e.g., extra saves and restores of data and registers, or for interrupts when at the edge of exceeding available time) can be taken care of.

j. Adequate growth potential exists for all function capacities (numbers of simultaneously trackable targets, etc.) or function limits (response time, etc.).

k. The volume of data each module can handle appears unlimited (e.g., sorts do not need to be revamped merely to accommodate bigger arrays).

5. The software is CORRECTABLE: Statements, parameters, and initializations are easily found, handled, and changed.

a. Program structure aids, not hinders, function insertions, deletions, and changes.

b. All parameters are identified, which could be modified without impacting system flow.

c. Changeable attributes (e.g., convergence) are named, and initialized in one place so that array sizes, I/O read-write unit numbers, arguments passed between routines, etc., utilize names which need only be changed once, where initialized centrally; this prevents having to change repeated fixed numbers in scattered locations.

d. All variables are centrally initialized; near or in the hierarchies that use them, in the case of local variables, or in a central data base for global variables.

e. Constants (local or global) which could change are so labeled, and are initialized in appropriate blocks that are easily found and easily changed by address or name.

f. Statements (executable instructions) are easily found and corrected.

6. The software is portable (SELF CONTAINED and DEVICE INDEPENDENT) and SUPPORTABLE.

a. While the source code (statements) is as much as possible in the High Order Language (HOL) of widest possible use and DOD-approved, it does not rely on uniquely-modified host compilers or assemblers, nor unproven host compilers, nor on obsolete languages or compilers falling into disuse.

b. At the time of acceptance, all software is totally available, in source language (without machine patches or manual intervention required), on media capable of being read and written on available agency computers.

c. Interfaces between target machine source and object language, and host processing, are simplified particularly in terms of minimizing host machine dependencies.

d. All reference routines not in the program are available on libraries (on peripherals, if necessary), in standardized formats and descriptions and are operationally usable "as is," and are in a form which can be copied and transferred to other machines, if necessary.

e. The instruction set is upward- and downward- compatible on this family of computers.

f. All assumptions about the operating system (I/O, interrupt levels, etc.) are stated.

g. Software organization permits installation in stages, if desired (e.g., first the Executive, then add I/O, then modes), but does not require it.

h. Machine dependencies (parity, unusual word sizes) are centrally explained, and all such dependencies are collected in centralized blocks; minimal mixing of applications functions occurs, with I/O, machine dependencies, etc.

i. Word-size-dependent operations (e.g., processing literal strings, numerical convergence) contain self checks against length incompatibilities.

j. Table values, in a form independent of machine types, drive most-used routines.

k. Any required source statements involving branches, transfer to labels, and relative branches (e.g., "down 17 statements") are not used in any cases that cannot satisfy the forward-only, to-to-less conventions, such as when low-level (assembly language) routines are unavoidably required.

l. The source statements are all traceable back to fully satisfy the design specs, and like the design specifications are as device-independent as possible.

m. The weapon system is easily "erasable" for a reload (not burned permanently into read-only memory, and not in microcode).

n. Software and hardware (though somewhat modified) were proven by previous use of this system or one of its family used in prior operational weapons systems.

## 7. SUPPORT RESOURCES are ADEQUATE.

a. Available host computer resources, especially if shared, include sufficient guaranteed time to update source, compile, simulate, and verify results.

b. Adequate support facilities for host computers are provided on host or resident gear. For example, adequate mass memory and programmer terminals are available.

c. Adequate host-supported utilities (sorts, splits, merges, prints, updates, file maintenance, debug packages, inter-device copies, etc.) are available to support any unique one-time or special processing which may be required occasionally.

d. Configuration control software is available on host or resident equipment. This is capable of identifying revisions of all software, protects against unauthorized updates or changes, and maintains approved and experimental software on host machines.

e. Adequate support software, written in High Order Language to the maximum extent, is operational on machines identical to those available to agencies tasked to maintain and support the weapons system software, and successful transfer to agency machines is guaranteed.

f. Support software can process all source statements into relocatable object modules.

g. Support software can link the selected relocatable object modules into an absolute executable load module.

h. Support software can generate on appropriate media an executable image which can be physically loaded into the weapons system.

i. Support software is capable of adequate simulation or emulation of the weapons system behavior, by execution of test programs based on the weapons system program.

j. Adequate test cases and procedures and expected results, or other verification means, are provided to demonstrate that support software does not cause misrepresentations during processing of source and object language.

k. Adequate test stands, and data reduction facilities are provided, to support testing of both support software and weapons system software.

l. Adequate training has been provided for, to help maintaining personnel identify and implement new system requirements involving recoding of source statements and patching of object language, and to help them identify, isolate, and fix problems and test their solutions.

C. Maintainable software is TESTABLE.

1. (Use also the criteria of I.A. (UNDERSTANDABLE STRUCTURE AND DESCRIPTION): Do not rescore.

2. Successive documents MAP correctly and completely to (and cross reference) each other, in order to allow testing.

Note: The Software Acceptance Criteria for external documentation (each document outside of the source code) would be a separate set of SAC for each document or set of documents which would constitute a different task. But the items below apply specifically to testability features of the software, which depend on portions of these external documents.

a. Each specification (spec) maps correctly and completely into successive specs.

(1) Thus, the system spec maps perfectly into the development (requirement) spec.

(2) The requirements spec maps perfectly into the test requirements/test plans spec.

(3) The requirements spec maps perfectly into the product (design) spec.

(4) The produce (design) spec traces perfectly into the CPC1 source code.

(5) The CPCI test plan maps or unfolds perfectly in the CPCI test procedures spec.

(6) The CPCI test procedures, when run, result in CPCI test reports (test fulfillment).

(7) The requirements and design specs are traceable into handbooks and manuals.

b. Each spec fully cross-references (by paragraph numbers) all specs to which it relates.

c. Each spec is hierarchically organized from most general concerns down to most detailed concerns, with numbering schemes for paragraphs, figures, etc., which correspond between successive specs to aid traceability and testability, and to facilitate changing, deleting, or adding functions and corresponding design and test of those functions in successive documents.

d. All parameters, beginning with requirements specs through design specs and code, have consistent nomenclature which immediately identifies each parameter clearly and unambiguously through various stages of fulfillment in successive specs to ensure I/O traceability and testability (e.g., 'lambda' in one doesn't become 'pv' in the succeeding spec); and the terminology is easily understood.

e. All off-the-shelf documents (previously developed for other uses) have been fully and accurately adapted to describe all idiosyncrasies of this weapons system; this ensures traceability and testability.

f. Specific information, especially that relating to testability, can be easily found within any document, by means of volume indices, tables of contents, glossaries, etc.

g. A master list of all documents pertaining in any way to this weapon system software is available, and briefly outlines relationships between documents in such a way as to aid traceability and testability of functions.

h. Each document contributes in its designated manner to assure these testable results:

(1) The CPCI (code) has been developed in accordance with proper requirements.

(2) The CPCI performs intended functions satisfactorily in the mission environment.

(3) The CPCI does not perform unintended functions.

i. Geometric drawings of vectors or other physical relationships adequately portray all such relationships where necessary, on separate pages in the appropriate specs to enable understanding of the process and interpretation of test results.

j. All sources (cited to page numbers or paragraph numbers of the reference) for all constants and algorithms are stated in the appropriate specs, in those cases where derivations are not provided in appendices, etc., so as to facilitate cross-checks of methods and results with other systems' results.

k. All intermodular relationships of any complexity are graphically portrayed by functional block diagrams, sequencing trees, timing and diagrams, displays, etc., in appropriate specifications, in testable terms and formats.

l. All documents are current (up-to-date descriptions of the latest requirements, design, code, tests, manuals, etc., which exist for this system).

m. All documents and code are deliverable in increments or builds as available.

n. No document or piece of code is vendor-proprietary (such that the Government cannot reproduce or disseminate it to other Government agencies or companies).

3. Requirements specs (including interface control documents or program requirements specs, and data base requirements specs) are testable and complete.

a. Requirements specs are organized hierarchically, from most general function down to the most detailed subfunction level, in accurate, complete, testable terms.

b. Requirements specs give full details of the technical and mission requirements and interfaces, all of which were allocated to this CPC1 by the system or system segment spec(s), in testable statements of sufficient accuracy and completeness to permit CPC1 testing and design to be defined based upon this document.

c. The requirements (development) spec contains a matrix of verification test methods, which shows how (by what type of test) each function and subfunction, down to the lowest level of requirements, will be satisfied.

d. For each subfunction, the method of verification appears to be adequate and appropriate, as stated in the requirements spec; nonverifications are justified.

e. Data base and data requirements are clearly stated in the CPC1 requirements spec, and conform to interface control documents which describe the relationships of data that are passed between other equipments, operators, software, or data bases.

f. For all input/output data elements, the requirements spec gives its attributes: data type, protection, usage (frequency) rates, quantity, range, limits, precision, accuracy, bit resolution, format, content, meaning, units, source or destination, transmission mode or channel, etc. These descriptions are given in a form which facilitates developing test plans, etc.

g. Each functional requirement is explicitly, quantitatively, and testably defined, in terms of inputs, processing, outputs, data requirements, data base, parameter definitions and symbols, output dependencies upon inputs, interfaces, required algorithms, accuracy, timing, exception handling, constraints, and pertinent performance.

h. An available error budget or accuracy control document helped drive the requirements spec, as evidenced by frequent cross-references between the two.

i. The requirements spec describes processing required to handle improper, incorrect, or out-of-range inputs (provisions for integrity), and resistance to noise (robustness) is provided for and described.

j. System functions and resource margins (e.g., timing and memory) are reasonably feasible at acceptable risk within allocated resources and schedules. (Unachievable requirements and designs make test planning and other routines difficult.)

k. Memory estimates, computation speed, I/O budgeting, and other resource utilizations are reasonable, self-consistent, feasible, and consistent with interface control documents and accuracy control specs, and hardware or systems specs, in resource allocations.

l. All requirements were clearly labeled as such (not confused with side comments), and concisely stated (without needless boilerplate).

m. Requirements specs have separate sections for external interface definitions, for defining each major program function, and for defining the program global data base requirements; major parts of the requirements specs are self contained.

n. Requirements statements for design, development, functional performance, and qualification testing are mutually consistent, complete, and reasonable.

o. Interface requirements statements take into account all types of interfaces:

(1) Software-to-hardware interfaces (e.g., sensor I/O, computer and control I/O, displays) are completely and accurately defined.

(2) Software-to-software interfaces (e.g., data transfers, computer execution control transfers) are completely and accurately defined.

(3) Software-to-personnel interfaces (inputs from operators, formats and medium of manual inputs, associated control, and data for display) are completely and accurately defined.

p. All requirements for design constraints are fully and accurately stated (requirements for safety and human performance requirements, specifications of programming language or limits on types of instructions used, memory allocation or protection limitations, self-test features (including recording for later diagnosis of faults), and processing rates required for external interfaces, etc.).

q. Requirements are stated for self-metric capabilities (e.g., continual recording of least-time left over from processing, nearness to data or memory saturation, recording of all faults by unambiguous flags for later diagnosis).

r. Requirements provide for maximum recovery from errors or loss of function, such that the system reverts to the next best usable mode, and such requirements are stated in testable form.

s. A requirements chart or matrix exists which accurately depicts interrelationships between requirements in such a fashion as to aid test planning.

t. Requirements are stated fully, accurately, and testably for startup/restart, initialization, executive control of hierarchied execution, including timing allocations, and exception handling (e.g., interrupts and errors).

u. Requirements are stated fully, accurately, and testably for backup modes, and to maintain best available modes in case of degradation or loss of function.

v. All accuracies and precision and bit resolution/scaling for all inputs, outputs, and algorithm and equation processing are stated in testable terms, and appear to be feasible and to provide the required accuracy throughout the requirements.

#### 4. Program design specs (product specs) enhance testability.

a. The design spec has an accurate and complete matrix that relates each subfunction (down to the lowest level) to each lowest level of subcomponent or component in the design spec that fulfills wholly or in part that subfunction, in order to enable traceability and testability continuity.

b. The design spec transforms the requirements into the exact configuration of the CPCI by detailing all inputs, outputs, interfaces, data bases, I/O formats, performance parameters, and flowcharts with complete, accurate, and testable statements.

c. The design spec provides complete and accurate descriptions in testable terms of each component, each subcomponent, and the data base.

d. The design spec describes the interrelationships and interactions of the program with the data base, and shows the interrelationships and interactions between subcomponents, in testable terms (e.g., dependencies, such that X executes before Y).

e. The design spec provides for appropriate execution rates and priorities of execution for various subcomponents, in order to properly fulfill requirements in an accurate, complete, and testable manner.

f. The design spec provides an adequate interrupt or task scheduler hierarchy to facilitate managing different external and internal demands in an orderly, hierarchied manner. Such exception handling also accommodates at least those errors that can cause error interrupts to occur. These priorities are fully stated in accurate, testable terms.

g. The Executive is determined in the design so that it oversees all transfers of control (except to certain common utilities), so as to facilitate testing; and redundancies (e.g., dual processors) that attempt to maintain the hierarchy and functions intact and recoverable, to protect against critical failures, both operational and testing.

h. The design spec provides for detection of (including notification to operators) and recovery from timing overrun (execution overload) in such a way as to preserve data that can later aid determination of the cause of such overloads, both in tests and in operational fleet fault detection.

i. All inputs and outputs listed in the design spec are accurate and testable, and are traceable back to the requirements spec(s) and satisfy all requirements.

j. All algorithms and equations in the design (product) spec are accurate and testable, are traceable back to the requirements spec, and satisfy all requirements.

k. Module interface inputs, outputs, order of arguments, and processing of inputs into outputs in the design spec correspond identically to the source code.

l. The design spec provides a separate section that describes a sufficient startup/restart and initialization process (both testable) for the weapons system.

m. The design spec provides a separate section that describes all exception (special) processing, including termination handling, handling external and internal errors (separately), and interrupts, all in testable terms.

n. The design spec provides separate sections specifying and explaining program interface inputs and (separately) outputs, in a complete, accurate, testable form.

o. For each module, the design spec has a separate description specifying and explaining each input, another for each output, and another explaining processing, all in terms which aid testing.

p. The design spec provides a complete and accurate diagram of overall functional control flow, including timing determinants for each major function, and the determinants (including timing) which determined priority execution order of modules, and their interruptibility, etc., all in testable terms.

q. The design spec contains an accurate, complete, testable, and easily understood set of charts depicting program flow (execution sequences of all modules); data flow hierarchies among all modules are also well depicted graphically.

r. For each module, the design spec provides detailed flowcharts or equivalents (depending upon language (structured or not) intended to implement the flow charts) that identify all possible branching, all processing, all I/O, all exceptions, etc.

s. The design spec provides testable memory storage estimates for each module or routine, and collectively for each major function, which generally fits any memory budgets specified previously in the requirements specs.

t. The design spec provides sufficient detail to enable coding from that design spec, and to verify (by equation reconstruction from the code, logic reconstruction of program flow from the code, I/O traces, and so on) that the code faithfully follows the design spec.

u. The format of the design spec closely follows the organization of the program, and each descriptive part tends to focus on one central concern.

v. The program timing scheme, as described in the design spec, is easily understood, appears flexible enough to allow for timing modifications for any module, and provides for reasonable timing margins for all functions, particularly those most likely to grow (so rate groups, time slices, priorities, etc., are adequate) and relates computer time lines with program time lines and windows.

w. The design spec establishes conventions to protect data integrity:

(1) The design spec requires the code to validate its own inputs (self-check tests, e.g., to substitute safe defaults for illegal array indices or bad inputs, and to record occurrences of these for later test and diagnosis).

(2) The design spec requires the code to detect, record, and where possible recover from internal failures or undefined operations—such as: underflow, overflow (such as from division by zero), attempts to take negative roots, or other illegal math arguments, singular matrices, divergence (instead of convergence) of algorithms, illegal operators or operands, to facilitate duplication and retest of such occurrences.

(3) Protection against wrongful data access (e.g., overwriting protected memory, attempts to access data not declared in the module to be required for access, illegal addresses, illegal I/O) is required of the code by the design spec and these occurrences are recorded to enable later retest and diagnosis.

(4) The design spec requires critical variables or register values to be saved (upon interrupts, entries to other routines, etc.) to prevent unplanned loss and vulnerability to interrupts, as well as recursive/re-entrant coding to be minimized to avoid excessive saves/restores and queue overloads.

(5) The design spec provides conventions for code to follow in module interfaces, and conventions for I/O processing (e.g., which modules do I/O, and of which type: random access/sequential/indexed/formatted or not) to aid testability.

x. The design spec establishes conventions for error processing to aid testing.

(1) The design spec predetermines which modules will perform error handling, what unique codes will uniquely identify each error, how to record or report such errors, and the format rules for any error messages to aid test monitoring.

(2) The results of any type of error or exception are explained in the design spec (e.g., what is the numerical result of dividing by zero, after recovery) in a way that aids analysis of test results.

y. The design spec has established protections against system lockups and other errors, but also provides information on recognizing and escaping from these if they occur, and records any such occurrences to aid later duplication for diagnosis and analysis.

z. The design spec provides for redundancies and backups in case of recoverable failures or degraded functions, in a way that can be testably recorded.

5. A set of test documents (test requirements, test plans, test procedures, test reports) demonstrates that the system is testable and able to function properly.

a. All requirements are traceable between the test requirements (often part of the requirements spec) and the test plan.

b. Test objectives in the test plan are clear and unambiguous and correspond exactly to the test requirements statements.

c. The test method type stated in the test plans agrees fully with the test method type for each function and subfunction that was stated in the requirements spec.

d. Test plans are fully adequate, either individually or collectively, to verify the function(s) or subfunction(s) they address.

e. All test dependencies are adequately defined and taken into account.

f. All tradeoffs and test limitations have been justified and explained adequately.

g. For each test, all pass/fail acceptance criteria are fully traceable back to statements of requirements in the requirements spec and are fully explained.

h. Test plans collectively have no needless redundancies that provide no more verification and make verification more diffuse and difficult.

i. Organizational responsibility is clearly defined for each planned test.

j. Resources and facilities are compatible and adequate to carry out the test plans.

k. Test schedules are achievable, even for substantial retests and problems in testing.

l. The test bed(s) have been previously validated, so that the weapons systems acceptance tests do not degenerate into a vehicle for validating the test bed(s).

m. The resources, facilities, and test tools are adequately described in the test plans and/or test procedures documents, and these describe limitations of these tools.

n. Any data models of expected results or acceptance criteria have themselves been previously validated or tested for completeness and accuracy.

o. The test plans, test procedures, and test results clearly identify the unique version of the end-item undergoing testing, and this may be verified by tape loads and other procedures.

p. Prior to testing, the weapon system software has been run through all applicable static analyzers, structure analyzers, test editors, code analyzers, symbolic evaluation systems, compilers, assemblers, link editors, flowcharters, etc., to determine that no known errors remain in syntax or logic, which could compromise or complicate the tests if not corrected; Appropriate listings of these results are available at the test site.

q. Test procedures provide complete step-by-step instructions of all steps needed to implement each test plan, and each test procedure corresponds exactly and is traceable to one or more test plans, which are completely and accurately fulfilled by the test procedure.

r. Test procedures provide adequate coverage of the test input space (nominal and extreme acceptable input data, and out-of-range data drive the test procedures), to exercise as much as possible of the code and logic paths.

s. Test plans and procedures documents, as determined by exhaustive reviews, have no mutual or self-inconsistencies or gaps in coverage, and are useful.

t. Test data requirements and data reduction techniques incorporated in the test procedures are complete and traceable to test plan acceptance criteria.

u. Test plans and procedures have made adequate provision to verify all interfaces (including hardware-to-software interfaces, software-to-software interfaces, and software-to-operator interfaces), including all hardware dependencies (tested preferably with nondestructive tests while performing required tests on failure detection, reporting, and recovery).

v. Provision exists in the test plans and procedures for maximum practical testing of all possible error conditions (both of internal and external origin), and particularly for all practical tests of possible hazard conditions.

w. Initialization tests, which verify correct and complete initialization of data, are provided for. These include storing test patterns in any data memory which must be reinitialized (overlaid) at each system turnon or new mission in order to assure that data left over from previous missions cannot cause errors, unpredictable results, or failures in the present mission for weapons systems which are "reusable."

x. Software tests were run in accordance with approved test procedures.

y. During tests, qualified witnesses certified that test procedures were followed under the stated conditions, and that the proper expected test results occurred; sufficient time was available during the tests for witnesses to be sure of the observations that they certified, and these witnesses included representatives from procuring or supporting agencies.

z. Current descriptions of the item to be tested, and its software printed on hardcopy listings, etc., test plans, and procedures, were available to witnesses sufficiently in advance of the tests for adequate review and familiarization.

aa. All test documents (plans, procedures, and reports) had provision for incremental delivery of sections as each became available.

bb. Test results met the minimum requirements stated in requirements specs to validate performance:

(1) A sufficient number of tests cases were run to validate performance.

(2) Boundary conditions are recoverable (tests are repeatable).

(3) Adequate analysis was performed and proper interpretation and evaluation was done.

(4) Tests results conform with requirements specs.

cc. All errors were identified and evaluated and corrective action to be taken was identified.

dd. All deficiencies were identified and evaluated, and corrective action to be taken was identified.

ee. Waivers and/or deviations were identified and documented, and sufficient rationale is available for evaluation.

ff. All test documents (test plans, test procedures, and test reports) conformed to applicable documentation standards.

## 6. MANAGEABLE DATA VOLUMES AND COMPLEXITIES characterize testable software.

a. Data rates are manageable (will not overwhelm testing facilities).

(1) Data transfer rates will not overwhelm instrumentation/reduction requirements.

(2) Data computation rates (cumulative, for all data which must be recorded serially, in-process, or on-the-fly) are not excessive (requiring complex sampling).

b. Algorithms for which intermediate results may require instrumentation in the code, are not so complex as to make difficult the recording of intermediate data.

c. The code does not involve great complexities (e.g., numerical algorithms, numerous logic paths and trees, complex timing synchronization, many interfaces that make test definitions difficult or complex).

d. Resource use (memory, timing, or interrupt excesses, or real-time conflicts or interference) can be recorded and interpreted.

e. Data organizations help definition of initial data values required for testing at each level, and consequent expected results, which were ideally placed in appropriate preface comments.

7. Code is WELL INSTRUMENTED, or test aids and probes can easily be added to test beds.

a. Checks (e.g., flags, probes, timers, performance monitors, in-range checks, type-checks, error-pickups) can be activated selectively and universally to detect, record, and where possible recover from anomalous conditions during tests.

b. Input and output data and intermediate (internal) results are recordable, from any desired routine, in a universal and flexibly selective manner.

c. Executive flow (trace of instruction sequences executed) is universally and selectively recordable for location (address) ranges, times, branches only, etc.

d. Coverage-of-test monitors can detect, for example, which instructions (addresses) were executed once or more, what branches or logic paths were taken.

e. Automatic universal and user-selective collection of saved/dumped test data enables real-time displays and/or post test data reduction and presentation, including user-specified and automatic data conversions of input/output data to man-readable forms (e.g., decimal numbers and graphics) for compression of results.

f. Support software (e.g., compilers, assembler, link edit loaders, simulators, tape generators, post processors) provide useful tools for debugging by detecting syntax errors, potential overflows, time excesses, etc.

g. All diagnostic or status messages provided during tests are clear, unique, unambiguous, and self-sufficient, and all erroneous inputs and processing errors are flagged.

h. All test aids embedded in the code are clearly identified and highlighted, and well explained.

i. Unchanged inputs need not be re-entered with each new test case execution; input is only required if a given parameter has changed since the last execution.

### Attachment III

Suggested changes in MIL-STD-1679, before its possible adaptation as a Tri-Service Standard:

1. Add to the end of Paragraph 1.2 this statement: "Any waivers of requirements stated herein shall also solicit the review of any using command and any supporting command or agency to which the developing agency will later transition the weapons system for maintenance and/or operational use, if these waivers of requirements might adversely affect system or subsystem maintainability or usability."

(Rationale for proposed change: If a requirement is proposed to be waived but that waiver might adversely affect the ability of using or supporting commands or agencies to use or support the system, they should have a chance later to at least point out how the waiver could degrade their capability to use or support the system later. The using and supporting commands will have to live with the consequences of any waivers or requirements in the operational life cycle, so the SPO should at least be reminded to consider their needs.)

2. MIL-STD-1679 generally specifies that it is the contractor's responsibility to perform this or that task. But some paragraphs do not specify who (the developing agency or the contractor, etc.) is responsible for the activities stated in the paragraph. The following paragraphs might leave some doubt as to whether it is the contractor's responsibility or the agency's: Paragraphs 4.3; 5.2.2.1; 5.2.2.2; 5.2.2.3 (and also subparagraphs a, b, and c); all paragraphs under (and including) 5.3 and 5.4; 5.5.3; 5.5.5; all under (and including) 5.5.6; 5.8.1 (including subparagraphs a, b, and c); 5.8.2 (including subparagraphs a thru 3); 5.8.3 (including subparagraphs a thru 3); 5.9.1; 5.9.1.4; 5.9.1.5; and possibly 5.11.2.2.

(Rationale for proposed change: particularly as in most instances the MIL-STD specifies whose responsibility it is (the contractor's or the agency's) to do a task, that performance should be uniform, and that each task should specify who is to do it or be responsible for it, i.e., the contractor or the procuring agency.)

3. As in the case of a minor type which could cause some problems because it is a paragraph number, between paragraphs 5.12.3.1 and 5.12.3.3, change the paragraph number from 5.11.3.2 to 5.12.3.2.

(Rationale: The paragraph number does not fit in sequence.)

4. Add to the last sentence of Paragraph 5.10.2.3 (which deals with software quality test documentation) statements requiring that the last incremental delivery of test documentation, or delivery of the full test document if not delivered in increments, shall be delivered to the testing activity and to the witnesses designated to represent the agency, soon enough to permit thorough review before testing commences. Also add that said witnesses shall certify that they have thoroughly reviewed the test documents before the test commences.

(Rationale: Witnesses must be well prepared to detect test deficiencies. Paragraphs 5.12.3.3 and 5.12.3.4 also require document deliveries soon enough to permit meaningful review.)

5. Paragraph 5.10.3.1 (b) says, "intermittent errors shall be included in the count of software errors and receive no special consideration". This implies that an intermittent error that occurs 9 times is only counted once, but that should be clarified as it would make a large difference if the error were counted 9 times instead of once.

(Rationale: Some potential ambiguity as to how to count intermittent errors (once? or as many occurrences as are found, since that is a "new" error each time it shows up after not being a problem for awhile) should be clarified to say that an identical intermittent error is counted only at its first occurrence, if that is the intent of this paragraph.)

6. Paragraph 5.1.2.6 (a) specifies description required for inputs and should include description of "meaning" to conform with Paragraph 5.1.2.6 (c), which specifies that "meaning" of each output shall be defined in the detailed functional requirements.

(Rationale: Descriptive requirements for inputs should be as great as for outputs, so "meaning" of inputs should also be described.)

7. The third line of Paragraph 5.4.4.1 (Abstracts) calls for a "list of other components called." This should say "a list of other components called, in order and number of calls."

(Rationale: Unless some indication is given as to how many calls to particular external routines are made, and in what order, readers of abstracts for components may not expect any calls to a particular routine after the first call is encountered by them in the source listing.)

8. Paragraph 5.4.5.3 prohibits compound or complex source statements except to support allowable control structures. This should be modified to refer only to control structures, as compound Boolean constructs, etc., might legitimately be used to set data values which had nothing to do with control flow at that point.

(Rationale: Some setting of data values, irrespective of control logic, might most understandably use compound statements to obtain the value. So long as it had no direct impact on control flow, that should not be prohibited. But Paragraph 5.4.5.3, unless modified to refer only to statements affecting the flow of control, might ban such methods of setting data, too. If the paragraph really intended to ban compound or complex statements which also had nothing to do with logic flow of control, as seems to result from the paragraph, that seems to have questionable justification for the ban.)

9. Paragraph 5.5.6.2 (Cross Reference Listing) should have something similar to the following addition: "This cross reference may be machine-generated, but to the extent that indexing, offsets (displacements), and multiple references or accesses to 2 or more operands in a series in a single instruction may cause missed or erroneous cross-reference entries, these occurrences shall be footnoted with correctionis." (The same footnote text may describe a number of identical errors, if all bear that same footnote number). Any coding which causes excessive cross-reference errors (by using index registers or other operand address displacements, or single instructions with multiple memory operand references, etc.) shall be minimized as much as possible. The cross-reference shall also flag each occurrence (line number or instruction address) which may set a variable (change its contents).

(Rationale: Cross-references, for reasons of economy and rapid reproductive capabilities, are generally machine-generated in connection with compilation or assembly or link edit runs which produce in-line, side-by-side listings, etc. Any instructions which contain indexing or operand reference displacements whose exact values may not be known to the software that generates the cross-reference may miss some actual operand references, and erroneously enter others. Such errors can be corrected by flagging and pointing to particular footnotes (which can often be transferred easily by cutting and pasting to new cross references). The excessive use of tricky indexing or other displacement-of-operand techniques causes many such cross-reference errors, as well as making the coding hard to follow.)

10. Add to Paragraph 5.8.1 a new subparagraph (d): "Ensure the capability of the module to handle properly and survive any erroneous external inputs."

(Rationale: Ideally, some higher level does I/O, but if any I/O to/from any external equipments is done in this routine, it should have a demonstrated capability to handle possible faulty inputs, so that undefined or unpredictable results do not occur.)

11. Paragraph 5.10.3.1 (c) (the number of unresolved technical errors in all deliverable documentation) is too stringent, even if there were a mutual understanding of whether a disputed statement was a "technical error," which may be too vague a term. One evaluator can find a large number of errors that can be fixed, and another with a different viewpoint or approach can find other errors. Also omitted was any mention of whether documentation errors (defined in Paragraph 3.9.2) include errors of omission, or only errors of commission (misstatements).

(Rationale: An evaluator who wants to "pass" the documentation can make the meaning of a "technical error" so stringent that even a New York City blackout would not suffice to demonstrate a "technical error." On the other hand, an overly critical reviewer can find no end of errors (quite aside from mere typos or misspellings) merely by going over the documents one more time, to "fail" the documents. The criteria is too vague, in the legal sense, in failing to give warning of what kind of statement in dispute might unquestionably be a "technical error," and what kind just as unquestionably would not be a technical error.)

12. Paragraph 4.2 states that "the design shall completely satisfy all requirements but shall not exceed the requirements without procuring agency approval." It might be better to say that "the design shall completely satisfy all requirements, but no unauthorized functions shall be provided without procuring agency approval."

(Rationale: The phrase "shall not exceed the requirements" almost seems to imply (notwithstanding the paragraphs which require at least 20% reserves of timing and memory) that no additional capacity can be pre-provided, but not used. It is easier and better to merely forbid unauthorized functions, which is not so vague a term as "exceed the requirements." Paragraph 5.10.2.7 dealing with stress testing, calls for requirements to be exceeded to assure that any exceedance of capacity does not result in catastrophic failure. But that clearly only refers to time and data handling exceedance of capacity, etc.)

13. The sentence of Paragraph 5.1.2.3 calls for contractors to identify all documents that define or constrain the program performance requirements. An additional clause might profitably be inserted in this sentence requiring them to identify the version numbers of those documents (that is almost, but not quite, implied). It should also require them to state which sets of paragraphs, etc., in each document constrain the program performance requirements.

(Rationale: Particularly where many documents of large size might be cited as constraining program performance requirements, the exact portions of those documents that are alleged to constrain the requirements should be identified. Otherwise, reviewers can have a difficult time wading through unnecessary material. Even more important, it helps define what those constraints are if we know which paragraph numbers (sets of paragraphs) impose the constraints. That way, if it is apparent that some other paragraphs in the same documents did not impose constraint, but seem to external reviewers to impose additional constraints, then such omissions can be noted and corrections or explanations requested.)

14. In Paragraph 5.10.2.9, the use of the term "off line" does not make totally clear whether that also includes support software run "off-line" on other machines, or if it only means maintenance/diagnostic programs, etc., which run on this machine, if it is not running the operational weapon system program. It probably means the latter, but that is uncertain.

(Rationale: As "off-line" software might mean two different things (stated above) to reviewers, the term should be clarified.)
15. Paragraph 5.4.6 says that "there is no requirement that flowcharts be a deliverable item." That may be too strong a statement. It might more justifiably say that flowcharts are not required unless specified as to extent and degree of detail by the procuring agency.

(Rationale: As in MIL-STD 1679's own figures 1a through 1f, depicting flowchart symbols for structured programs, flowcharts of various types may be helpful, so should not be dismissed without consideration of their value. It is true that structured programming, by avoiding "go to 's'", etc., need not rely nearly so much on flowcharts to show control flow if it is top to bottom and indented in the code. But the agency, for the sake of efficiency, may allow a developer to use certain assembly or machine code routines which are run so often that they must be optimized. If the blanket statement remains that no flowcharts are required, this would extend also to lower level assembly routines where flowcharts are most needed. Further, generation of source language, in many programming/documentation standards, is to be done from flowcharts; if these are absent, so is the documented basis for generating the source itself. Finally, paragraph 5.2.2.3 requires a determination of flow of program data and control in all required modes. That might, in many cases, require flowcharts.)
16. Paragraph 5.4.1 says, "symbolic parameters shall be grouped at the beginning of each subprogram." The intent is generally to take any symbolic parameters in any block of code, and put them at the front. But "subprogram" too narrowly may imply only a certain hierarchical level of organization, when in fact symbolic parameters may be found in "subprograms, routines, modules, and procedures," depending upon the system and application. This latter set of four might better replace the single word "subprogram."

(Rationale: The possible restriction to only one hierarchical level (subprogram) might be too narrowly interpreted to mean exclusion of grouping in other hierarchical levels. The definitions in Paragraph 3.3 (component) are clearly more applicable, i.e., not restricted to just one hierarchical level.)
17. In Paragraph 5.3.6, "Recursive" may need explanation, or perhaps it is explained in the glossary (AD A056868). At it is the subject of a complete paragraph, however, perhaps "recursive" should be defined in Section 3 (definitions.)

(Rationale: To some people, "recursive" procedures or routines are those which call themselves. To others, it implies merely a special kind of looping. Perhaps others have different definitions. One definition should be chosen and given here.)

18. In Paragraph 5.1.3 (system resources), perhaps the parenthetical expression "(including size)" should be added after "memory."

(Rationale: Definition of "computer memory" might be taken to mean merely a definition of types of memory structure, etc. But the requirement should clearly specify that the amount or type of memory will be explained.

19. Paragraph 5.8.2 (d) might be changed to replace "ensure the" with the statement "ensure the adequacy of."

(Rationale: "ensure the" implies making sure that the interfaces exist, but says nothing about their adequacy.

20. In Paragraphs 5.1.2.5 (c) and 5.2.3 (virtually identical paragraphs describing "Intersystem interface"), add to the sentence which brings "data quantity, frequency, rate, format, content, scaling requirements, and conventions shall be developed", the following words, inserted after the work "requirements": "source and destination, type, range, and limits, accuracy requirements, meaning of each data element."

(Rationale: The intersystem interface should specify which system sends what data ("meaning of each data element") to what other systems ("source and destination"). Further, the type of data (discrete, continuous variable, floating point or fixed, single or double precision, etc.) needs to be known. Range and limits for intersystem variables partly determine what each system must do with the data, so they should be specified here. The same is true of accuracy requirements which is more than just the value of the least significant bit, but includes algorithms error, etc.)

21. In Paragraph 5.1.2.6 (a) (inputs), add "meaning" (which was specified in Paragraph 5.1.2.6 (c) for outputs, so is equally necessary for inputs) to the list of input attributes. Also add "type and accuracy required" and change "range" to "range and limits."

(Rationale: "Meaning" should as much describe inputs as outputs. And the "type" of data (discrete, continuous variable, floating point or fixed, single or double precision: single word or doubleword) is also needed for interface definitions. By the same token, "accuracy required" must be known to establish how much care must be taken in processing such inputs, in terms of truncation, called out in Paragraph 5.4.3.4, which addressed the number of significant digits in inputs. "Range" may or may not be synonymous with "limits," as a normal range of a variable might have a separate set of "limits" imposed beyond that range for purposes of safety, protection against machine exceptions (divide by zero, etc.) Thus, "range and limits" is more complete and covers the minority of cases where they are not the same thing.)

22. In Paragraph 5.1.2.6 (c) (outputs), and "quantity" (which was specified in Paragraph 5.2.1.6 (a) for inputs, and is equally necessary for outputs). Also add "type and accuracy required" and change "range" to "range and limits."

(Rationale: For "quantity," that is as useful for outputs as for inputs. For "type," the same rationale as for input types exists. For "accuracy required," this requirement determine the degree of algorithmic or computational accuracy that will be required of the outputs, which dictates whether single or doubleword calculations, etc., will be used. The same rationale exists for changing "range" to "range and limits" as was given in the rationale for changing "Paragraph 5.1.2.6 (a) (inputs).)

23. The only reference to required or permissible cycle times for parts of the software subsystem is found in Paragraph 5.2.2.3 (b), which addresses permissible cycle times for each subprogram. "Subprogram" should perhaps be changed to "component" so that permissible cycle times are not restricted only to subprogram levels, in terms of requirements, definitions. By the same token, Paragraph 5.2.2.3 itself should perhaps change "subprogram" to component," where it presently requires a functional description of all inputs, outputs, and processing only for each subprogram.

(Rationale: Paragraph 3.3's definition of subprogram equates it to a major functional subset of a program, and says it's made up of one or more modules. The same definition Paragraph (3.3) defines a module as an independently compilable software component, comprised of one or more procedures and routines, which is the lowest level defined by Paragraph 3.3. A component is defined as a subset of the weapon system software which can be hierarchically broken down into components of program, subprogram, module, and procedure or routine. Thus, Paragraphs 5.2.2.3 and 5.2.2.3 (b) only require definition of permissible cycle times, and inputs, outputs, and processing seemingly at the subprogram level. But there should be a requirement to define permissible cycle times, and inputs, outputs, and processing for all components, not merely at the subprogram level of hierarchy. Paragraph 5.4.4.1 states a requirement to list all other "components" called, and all calling components. The business of calling implies (permissible) cycle times, and inputs, outputs, and processing requirements, all of which can have to do with any component, not just the subprogram level of components.)

24. Paragraph 5.2.2.4 addresses resource allocation and reserves, including memory reserces, stating that "total system memory, input and output channels, and processing time reserves of at least 20 percent shall exist at the time of program acceptance..." This should be changed to say "total system memory of each unique type not interchangeable with other memory types existing in the weapons system shall have reserves of at least 20 percent at the time of program acceptance..." The same kind of change is needed for the 20 percent reserve requirement for input and output channels of unique types not interchangeable with other I/O channels which help comprise the weapons system. The same is true for definitions of possible different types of processing time reserves of 20 percent or more; this should say processing time of each unique type or independent embedded computer. The same type of changes should be made in Paragraphs 5.5.2 and 5.10.1.

(Rationale: If several kinds of memory exist in the embedded computer (sub)system, such as RAM and RCM memory, or memory of different lengths (such as 15 bit memory in one section, and 19 bit memory in another), and these are not interchangeable, the system growth may be stymied even if a "total" system memory reserve of 20 percent exists. This can occur if, for example, plenty of RAM memory exists to produce a total reserve of 20 percent, but no ROM instruction memory reserve exists for growth of program instructions. Or if plenty of ROM constant and instruction memory exists, but no more RAM variables can be accommodated, even though total system memory technically has over 20 percent of total system memory reserves, again system growth is stymied. Any memory type which cannot be substituted for other memory types now on the weapons (sub)system is thus vulnerable to this problem. The same difficulty exists for I/O channels of different, noninterchangeable types, and for multiprocessor types which are not interchangeable in the functional sense, so that one runs out of time while the other has 50 percent left.

25. In Paragraph 5.2.2.6, data base design, the contractor is to take into account all data used by two or more subprograms. Perhaps this should be changed to say two or more components (not subprograms).

(Rationale: A subprogram can consist of one or more modules, which in turn can consist of one or more procedures or routines, by Paragraph 3.3's definitions. But suppose that 27 routines communicate between themselves as part of the same module, or that 13 modules pass data between each other, but all are part of the same subprogram. Then there would be no requirement for the contractor to take this data into account, merely because they happened to be in the same subprogram. In fact, such hierarchical grouping could be used (all in the same subprogram) to avoid this requirement. It may even be true that most data is passed between two routines or modules within the area subprogram, in which case most of the data could be excluded from the data base design. The requirements for data base design should not depend upon arbitrary allocations of modules or functions into (or not into) subprograms.)

26. In Paragraph 5.4.3.2, the term "mixed mode" is used without having been defined in Section 3's definitions. It might mean such things as multiplying integer by floating point numbers, or might be broader to include such operations as adding single word data to double word data, or multiplying such different length words together, etc. Perhaps the "AVSO ADP Glossary (NAVSO-P-3097) mentioned in Section 2 has such a definition, but it may not be readily available.

(Rationale: As terms such as "patch" and "shall" etc., are defined in Section 3 or within the text that uses such words, so also "mixed mode" should be defined to avoid misunderstanding of this commenting requirement.)

27. Paragraph 5.4.3.4 refers to significant digits of outputs, and says that "the degree of computational error shall be analyzed to determine if systems accuracy requirements are fulfilled." But nowhere, it seems, is there any explicit statement of requirements to define accuracy requirements. This could be done as suggested earlier, in Paragraphs 5.2.2.5 (c), 5.2.3, 5.1.2.6 (a), 5.1.2.6 (c), etc. But somewhere a requirement for statement of accuracy requirements is necessary to the meaning of Paragraph 5.4.3.4.

(Rationale: If no requirement exists to state all input and particularly output accuracies somewhere, there is nothing for "significant digit" requirements to be compared or checked against.)
28. Add to the first sentence of Paragraph 5.4.4.1 "any unusual exits or terminations."
29. Paragraphs 5.5.5 and 5.11.1.2 refer to sequence numbering of the smallest independently compilable units of code (modules). Paragraph 5.5.5 requires sequence numbering in some multiple of tens; Paragraph 5.11.1.2 provides for structuring sequent numbering so that "future changes to any component can be properly noted." The latter almost implies that the least significant digit(s), which would originally be 0's for multiples of tens in sequence numbering, would be used for noting future changes. (If the most significant digits of sequence numbers were used to denote changes, that would compromise the sequence numbering itself). Thus, Paragraph 5.11.1.2 should perhaps state that part of the sequence number field shall be used exclusively as a means of identifying update numbers or revisions, etc., and that this shall be the least significant digit(s) of the sequence number field.

(Rationale: Though this may be a minor point, sequence numbers are often used for more than just sequencing. This most frequently includes using the least significant (rightmost) digits to record version number or update number. Some management systems, such as PAMVALET or LIBRARIAN require certain reservations as part of the configuration management system (say, three digits) to identify updates in the order in which they occurred in the source program.)
30. Add to the last sentence of Paragraph 5.7 (procuring agency approval of procedures for operating weapons system software) these words, "which shall also submit the procedures to all agencies scheduled to use the system operationally, for those agencies' review."

(Rationale: As operating procedures will be the direct concern of the agencies which will operationally use the weapons system, their comments and perhaps their concurrence should help shape those procedures. Using agencies will have to live with the procedures for the better part of the weapons systems' life cycle.)

31. Add to the end of the first sentence of the second paragraph of Paragraph 5.3 the following, "The procuring agency shall have the option of requiring incremental delivery of test specifications, or of test plans, or of test procedures, or of test reports, or of all of these, as they are developed in increments.

(Rationale: These are generally developed in increments, and so should be available for review on a timely bases as they are developed. This avoids the problem of having to review massive test-associated documents all at once, often with very little time. Comments can be returned more quickly and incorporated for revisions even before some of the later sections of the test documents are produced.)

32. Add after the end of the second sentence of the second paragraph of Paragraph 5.8 the following, "Such representatives may include representatives of agencies scheduled to assume maintenance or operation of the system after it becomes operational. All witnesses designated by the procuring agency shall be qualified by adequate familiarity with the system.

(Rationale: Witnesses should be capable of determining the validity of tests. This requires considerable familiarity with the system. The agencies in line to use or maintain the system after it becomes operational have bested interests in getting maintainable and usable systems, so representatives from these agencies can advantageously supplement those of the procuring agency, particularly if they participated in previous analyses and tests.)

33. Add the following after the end of the third sentence of the second paragraph of Paragraph 5.8, "Descriptions of the same shall be provided soon enough before the tests to enable meaningful review by the procuring agency."

(Rationale: This merely reinforces the provisions of Paragraph 5.12.3.3, which provides for preliminary working level reviews of all documents followed by formal reviews, delivered "sufficiently in advance" to allow adequate internal review by each activity.)

34. Add the following to the end of the second phase of Paragraph 5.12.3.3; "All system requirements shall be stated, or explicit paragraph references in deliverable documents shall be cited for each system requirement. Methods of fulfillment of testing requirements for each system requirement will be stated."

(Rationale: Some single test document should collect all system requirements, or references in deliverable documents to all system requirements, or both. Otherwise, it is difficult to determine even what is to be tested. It is not uncommon for many system "requirements" to be slipped (by reason of schedule exigencies, etc.) from requirements or development specifications, into design or product documents. But if testing is done against the paragraphs in the requirements documents, those requirements that were "slipped" into the design documents may not show up in the tests. It is far more sure to require that some one test document contain all testable requirements than to require that tests be based purely upon the requirements document(s) which does not always contain all requirements.)

35. Add to Paragraph 3.9.1 (software error definition) these words, "or causes unauthorized or harmful results."

(Rationale: The statement of a software error presently implies failure to achieve a specified performance, but does not really explicitly address undesired effects which were not intended.)

36. In Paragraph 5.8.1, add 3 new lines: (d) "ensure capability of the module to detect, handle properly, and survive erroneous inputs"; (e) "exercise all logic paths in the module"; (f) "exercise all algorithms with inputs, or representative values and limit-exceeding (data) values."

(Rationale: Any input which might be erroneous (particularly from uncontrollable external sources) should be so tested in modules which use it to verify the adequacy of protective or recovery measures. An unless all logic paths, including conditional branching, can be checked, the module has not had a complete check of its instructions. (This does not imply that all possible sets of data values need be tested.) Further, algorithms must be at least nominally tested, which at a minimum means testing with a nominal input(s) and one each of any group of extreme values at and beyond its design limits.)

37. To Paragraph 5.10.2.4, add the following, "where possible, nondestructive testing of software failure detection/recovery shall be used, but all failure detection capabilities shall be tested."

(Rationale: Any failure modes which are not tested may later emerge during abnormal conditions, which not only might constitute a safety problem, but also make observation difficult for subsequent ground-based debugging. And failures which can only be done by destructive testing are more difficult to repeat, if only from the logistics standpoint of time required to replace a destroyed part upon which failure test relies for retest.)

38. To Paragraph 5.6, add the following, "Delivery on physical media, machine readable on government equipment, shall consist of both source and object programs."

(Rationale: Typically, weapons system software source programs are transformed by compilers, assemblers, translators, etc., into the object language on off-line, large-scale general purpose computers. But at most installations, installation uniqueness exist, even on "identical" machines. This stems from the fact that on any given installation which supports standard language processors, of the up to thousands of patches which might be incorporated as "fixes," no large installation has incorporated all of them, and so rarely will patch sets be identical between even "identical" large computers used at a development facility versus a government facility to be used for software maintenance. This always raises some suspicions that perhaps the transformation from source to object will not be identical, particularly if unusual language extensions are used in the compilers, etc. Only if the object is supplied as a second file for the source from which it was produced can the receiving installation check on a record-by-record basis to be sure that the source was correctly transformed into object.)

39. To Paragraph 5.8.3 (b), add the following, "Initialization tests shall include storing test patterns in any data memory which is required to be re-initialized (overlaid) at each system turnon or new mission. This will assure that any data left over from previous missions cannot cause unpredictable results, errors, or failures in the present mission, for weapons systems which are "reusable."

(Rationale: Unless some means is provided to be sure that all initialization causes a positive overlay or overwrite of the particular memory required to be initialized, the possibility exists of "old" data in certain rare combinations causing problems for "new" missions.)

40. Add to Paragraph 5.10.2.1 (test environment) these words, "All 'truth models' and test tools, including interfaces, shall have been verified and validated for their use in these tests before the system test begins."

(Rationale: If the test tools are not prevalidated, the danger exists that the test will become more a vehicle for validating the test tools than for validating the weapons system software.)

41. To Paragraph 5.10.3.2 (patch limits), add, "In general, conditional branching logic, etc., should not be patched over for testing. Instead, data values should be patched or generated by execution to cause the conditioned branches needed for the tests. If such conditionaly branches must be patched over, other tests will check those unpatched conditional branches."

(Rationale: If tests patch over conditional branches, they cannot test, at least for that test, the validity of the conditioned branching and, therefore, any instruction it might access for those data values.)

42. In Paragraph 5.8.3 (a), add these words after the word 'ensure', "the workability of," so it reads "ensure the workability of the total man-machine interface."  
  
(Rationale: "Ensure the man-machine interface is an action empty of meaning unless it is stated what about the man-machine interface will be ensured.)
43. To Paragraph 5.10.2.5, add these words, "Sufficient time shall be allocated to each test to enable the agency-designated witnesses to adequately verify all initializations, switch settings, operator procedures, and to record all observable results."  
  
(Rationale: Test operators can become so proficient that the rapidity of their finger movements could make Blackstone the Magician envious. Then witnesses simply cannot follow and verify all of their actions in setting toggle switches, and observing results (often in the form of binary numbers displayed in rows of lamps). The witnesses cannot then really "witness" the tests adequately. This is usually the result of the operators' management (or the operators) failing to allocate enough time to go through the tests sufficiently slowly to permit adequate observation by the witnesses.)
44. To Paragraph 5.10.2.6, add the following, "All tests shall be in random order, except that those software portions (e.g., data entry and display, etc.) upon which other tests may depend, shall be done first."  
  
(Rationale: If certain portions of the software test involving software upon which other tests depend are not done first, it calls into question those dependent test results. Particularly if a failure occurs, it then becomes difficult to determine if the fault lies in the software being tested directly now, or in the software which is being used indirectly (as for data entry and display, etc.). To the extent that software should build upon that software upon which it depends by first validating the latter, the present statement that tests will be in random order decreases somewhat the confidence in the results. Other than the dependency problem, it is indeed better to have the tests run in random order, to detect any inadvertent test order dependencies.)
45. Change Paragraph 5.10.3.1 (d), "all software errors," to say "all software and documentation errors."  
  
(Rationale: Because Paragraphs 3.9.1 and 3.9.2 specifically distinguish between software errors and documentation errors, respectively, the paragraph as it now stands fails to state a requirement to document errors discovered in the documentation during the test run. Documentation errors should indeed be distinguished from software errors. But the provisions of Paragraph 5.10.3.1 (c) (allowable errors in deliverable documentation) should certainly include any documentation errors found during the test run, and Paragraph 5.10.3.1 (d), if not modified, fails to provide for this.)

46. Change the phrase in Paragraph 5.11 (a) "positive identification" to "positive identification and description."

(Rationale: Identification is incomplete if not coupled with at least a small amount of description for configuration management purposes.)

47. Add a new Paragraph 5.10.2.11, as follows, "The procuring agency will designate a qualified representative, which may be from the activity scheduled to maintain and update the software to test the adequacy of the contractor-deliverable maintenance documents and software for agency use. In this test, the qualified agency representative will verify on the appropriate machines that the deliverables are adequate by themselves to enable complete updates of the source and object programs of the weapons system. This test will perform a simple source update involving two or more different modules. Likewise a simple object update (patch) involving two or more separate modules shall be done. This will provide a user-oriented proof of the adequacy of software update procedures. If no such computer or software or hardware are to be located at any Government activity, the appropriate demonstration of update capability will be done at the contractor's development site, and will be done by contractor personnel in that event, but subject to agency approval.

(Rationale: Some live test of update capability (involving also linking together two or more separate sections of code) is needed to assure that such deliverables are usable by users (maintainers) directly from self-sufficient documents and supplied software and software procedures. Otherwise, uncertainty exists about whether nondevelopers can update code.)

#### Suggested Changes to MIL-STD-483:

1. Change "accuracy/precision" to "accuracy, precision" in Paragraphs 60.4.3.2.1.1 (inputs), 60.4.3.2.1.3 (outputs), 60.4.3.3.3 (system parameters) and any other places in MIL-STD-483 where such a phrase may occur. This will prevent misinterpretation of this clause by contractors to mean "accuracy or precision," in the sense of merely supplying bit precision of inputs and outputs instead of actual accuracy in the sense of algorithm error, etc.

(Rationale: The slash (/) is ambiguous. It may erroneously be taken to allow the contractor to merely specify the value of the least significant bit (scaling) of an input or output, instead of the algorithmic and bit error, as determined by error budget analysis, etc. If only the scale value of the least significant bit is given, this means that meaningful analysis of accuracy is almost impossible.

2. In MIL-STD-490, Paragraph 60.4.3, acceptance test requirements were specified to be accomplished, but in MIL-STD-483, Paragraph 60.4.4.3, which was supposed to supplement and follow 490 (a DOD document), the Air Force 483 said that acceptance test requirements were not applicable to CPCIs. That has the effect of repealing MIL-STD-490's Paragraph 60.4.3, yet MIL-STD-483 should comply with it.

(Rationale: If DOD 490 has precedence over Air Force MIL-STD-483, then MIL-STD-483 should conform and not contradict MIL-STD-490.)

